البرمجة المرئية بلغة Visual C#.Net2008

إعداد /

أ. محمود عثمان يحي مبارك

الفصل الأول

مقدمة في لغة C# &حزمة Visual studio.Net

- مقدمة :

منذ بداية تاريخ ظهور الحاسبات الإلكترونية دعت الحاجة إلى ظهور لغات برمجة بواسطتها يتم إنتاج برمجيات لتشغيل هذه الحاسبات وتطبيقات برمجية تساعد في حل مشكلات المستخدمين في شتى مجالات الحياة التي يستخدم فيها الحاسوب لغرض معالجة البيانات بمختلف مستوياتها والحصول على نتائج ومعلومات دقيقة بأشكال مختلفة تلبي حاجات المستخدمين وترضي طموحاتهم وتطلعاتهم .
وقد تزامن تطور لغات البرمجة بشكل سريع جداً يوازي تطور الحاسبات الإلكترونية بأنواعها والأجهزة الإلكترونية التي تحتاج لأنظمة تشغيل وتعمل عليها برامج تطبيقية مثل الهواتف الذكية وغيرها .
وكلما كانت إمكانيات لغات البرمجة عالية الأداء وذات أمنية قوية وسهلة الإستخدام ومختصرة الأكواد ومواكبة للتزامن التكنولوجي والمعلوماتي أنتجت هذه اللغات أنظمة تشغيل برامج تطبيقية عالية الجودة وتلبي رغبات وحاجات المستخدمين ... من أفضل وأروع هذه اللغات لغة C# .

- نبذة مختصرة عن لغات البرمجة :

يمكن تصنيف لغات البرمجة إلى ثلاثة أنواع :

- لغات برمجة متدنية المستوى : وتتمثل في لغة الآلة (0,1) .
- لغات برمجة متوسطة المستوى : وتمثل في لغة التجميع Assembler وكذلك المترجمات Compiler والمفسرات Interpreter الخاصة بلغات البرمجة عالية المستوى .
- لغات البرمجة عالية المستوى : وهي لغات البرمجة التي أوامرها وإجراءاتها قريبة من لغة الإنسان ويمكن تصنيفها حسب مجال التطبيقات التي تنتجها مثلاً :
  //لغات البرمجة الإجرائية مثل Q Basic ، C ، C++ ، ...
  // لغات البرمجة كائنية التوجه مثل Java ، C# ، ...
  // لغات برمجة قواعد البيانات مثل Oracle ، SQL Server ، ...
  // لغات البرمجة المرئية مثل Visual Basic.Net ، Visual C#.Net ، ...
  // لغات تصميم مواقع الإنترنت مثل HTML ، PHP ، ASP.Net ، ...
  وغيرها من مجالات برمجة الحاسوب .

- تاريخ لغة C# :

ظهرت لغة C# بشكل منتج برمجيتقريباً خلال عام 2001م وقد تبنتها شركة Microsoft وساهمت في تطويرها ضمن منتج حزمة Visual Studio.Net مع عدة لغات آخرى بحيث أدمجت في منتج VS.Net2005 ، VS.Net2008 ، VS.Net2010 ، VS.Net2012 ، ... .

وتعتبر لغة Visual C#.Net من لغات البرمجة متعددة الأغراض بحيث يمكن بواسطتها إنتاج تطبيقات برمجية في أكثر من مجال ويمكن أن نذكر أهم التطبيقات البرمجية التي تنتجها VC#.Net كالتالي :

١- Console Application : ويتم فيه عمل تطبيقات خاصة بالبرمجة الإجرائية والبرمجة كائنية التوجهOOP وتظهر النتائج البرمجية في هذا التطبيق على شاشة الـ DOS .
٢- Windows Form Application : ويتم فيها عمل تطبيقات خاصة بالبرمجة المرئية وتظهر النتائج البرمجية في هذا التطبيق على شكل نوافذ تشبه تلك الموجودة في نظام التشغيل Windows .
٣- (Web)ASP.Net Application : ويتم فيها تصميم وبرمجة مواقع وصفحات الويب بشكل ديناميكي حديث بحيث أصبح بالإمكان تصميم وبرمجة قواعد البيانات وعمل تطبيقات مرئية ونوافذ على صفحات الويب بمنتج واحد .

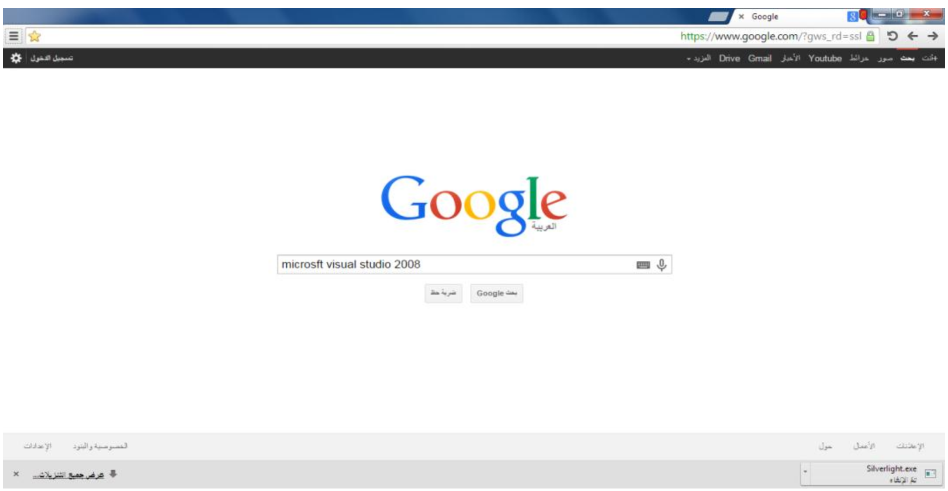
وتوجد تطبيقات أخرى مثل WPF ، WCF وتطبيقات الـOffice ، DataBase، Reporting، Smart Deviceوغيرها من التطبيقات البرمجية الهامة .

- حزمة Microsoft Visual Studio.Net البرمجية :

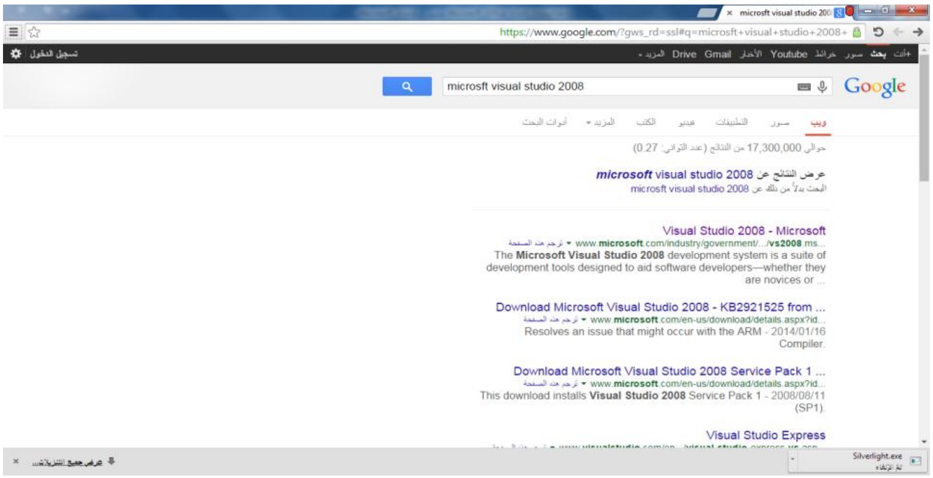
وهي المنتج الرسمي للغات البرمجة متعددة الأغراض لـ شركة Microsoft وتضم عدة لغات برمجة أهمها : (VB)Visual Basic.Net– (VC#)Visual C#.Net– (VC++)Visual C++.Net ... إلخ .

ويمكن تحميل هذه الحزمة البرمجية من شبكة الإنترنت عبر موقع Microsoft ، إتبع الخطوات التالية :

- إفتح برنامج مستعرض الويب الموجود في جهازك ثم قم بفتح محرك البحث Google عبر كتابة العنوان www.google.com .
- إكتب في مربع بحث google: "Microsoft Visual Studio 2008"



- تظهر صفحة النتائج ومن ضمنها Microsoft – Visual Studio 2008كالتالي :



- إنقر على الرابط Microsoft – Visual Studio 2008 تظهر الصفحة التالية :



توجد النسخة الكاملة من Microsoft Visual Studio 2008 وتوجد ثلاثة أزرار :

- Try Now : لتحميل نسخة تجريبية من المنتج .
- Buy Now : لشراء نسخة أصلية من المنتج .
- Contact Us : إتصل بنا للتواصل مع إدارة موقع Microsoft حول المنتج .

إتبع التعليمات بعد النقر على أحد الأزرار الثلاثة .

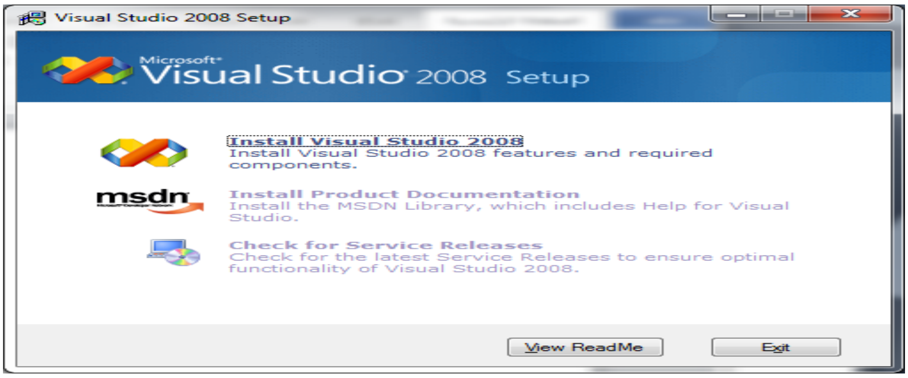
// ملاحظة : يمكن الحصول على نسخة من المنتج على قرص DVD من محلات بيع البرامج في منطقتك مع العلم أن شركة Microsoft أنتجت أكثر من نسخة من المنتج مثلاً كالتالي :

Team , Professional , Express , ...

- خطوات تنصيب حزمة Microsoft Visual Studio 2008 & مكتبة التعليمات MSDN:

- ضع قرص الـDVD للحزمة في CD\DVD Driver وقم بعمل قراءة تلقائية أو من جهاز الكمبيوتر –>إفتح قرص CD\DVD <- إنقر نقراً مزدوجاً على الملف Setup 

- تظهر نافذة Visual Studio 2008 Setup يوجد بها ثلاثة خيارات كالتالي :

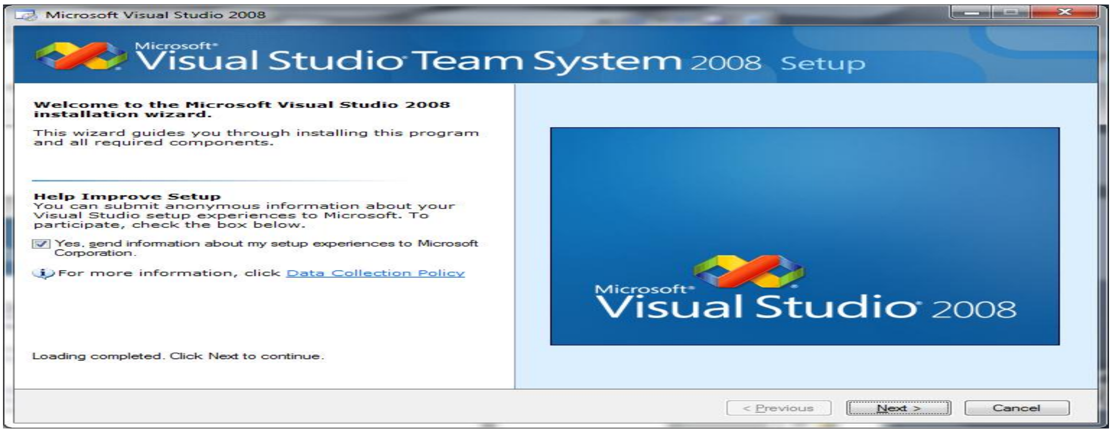


= Install Visual Studio 2008 : بالنقر عليه يبدأ في خطوات تنصيب اللغة .

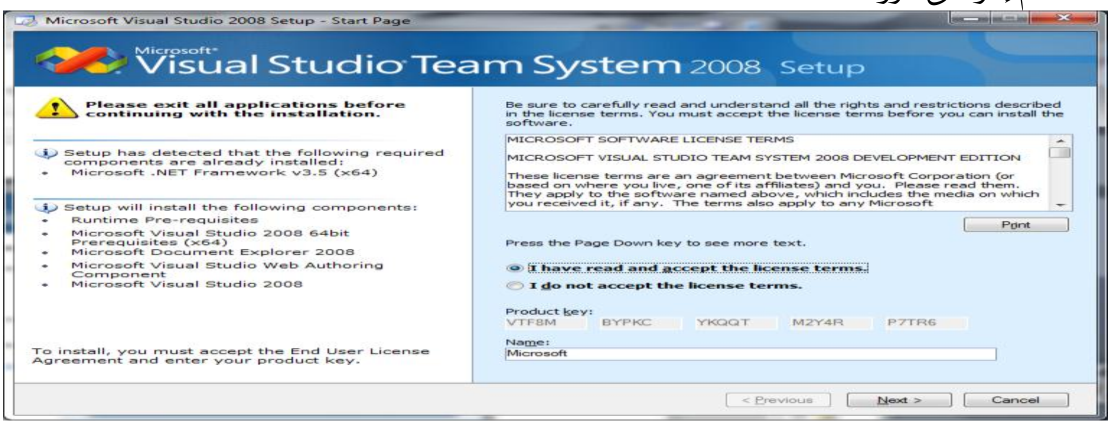
= Install Product Documentation: بالنقر عليه يقوم بتنصيب مكتبة التعليمات MSDN Library التي توفر صفحات مساعدة Help للمبرمج .

= Check for Services Releases : بالنقر عليه يقوم بالتحقق من وجود إصدارات الخدمات التي تقدمها VS.Net . إنقر على الخيار الأول للبدء بعملية التنصيب .

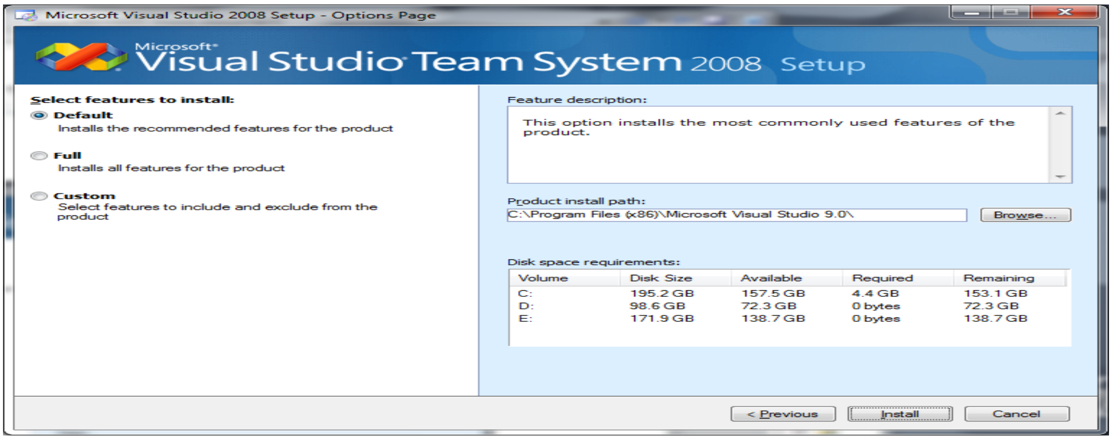
- إنتظرقليلاًحتى يتم نسخ ملفات التنصيب بعدها تظهر نافذة ترحيبية بالمستخدم حول تنصيب الحزمة إنتظر حتى نهاية التحميل ثم إنقر الزر Next .



تظهر نافذة حول الإتفاقية وشروط الشركة المنتجة يوجد خياران : حدد الخيار الأول I have read and accept the license terms دلالة على الموافقة على الإتفاقية ثم إنقر على الزر Next .



- تظهر نافذة خيارات التثبيت توجد ثلاثة خيارات كالتالي :



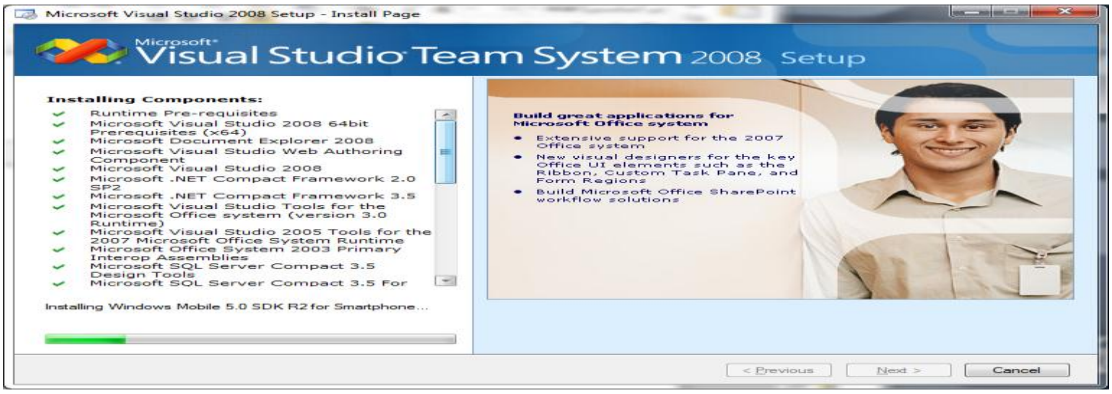
= Default (إفتراضي) : ويتم تثبيت البرامج الضرورية من حزمة Visual Studio 2008 والمحددة من قبل الشركة المنتجة وهو الخيار الأفضل .

= Full (كامل) : ويتم تثبيت كافة برامج حزمة Visual Studio 2008 .

= Custom (مخصص) : وفي هذه الحالة يقوم المستخدم بإختيار البرامج التي يريد تثبيتها من الحزمة حسب رغباته ونحتاج في هذه الحالة إلى مبرمج محترف وملم بعمل كل البرامج في الحزمة .

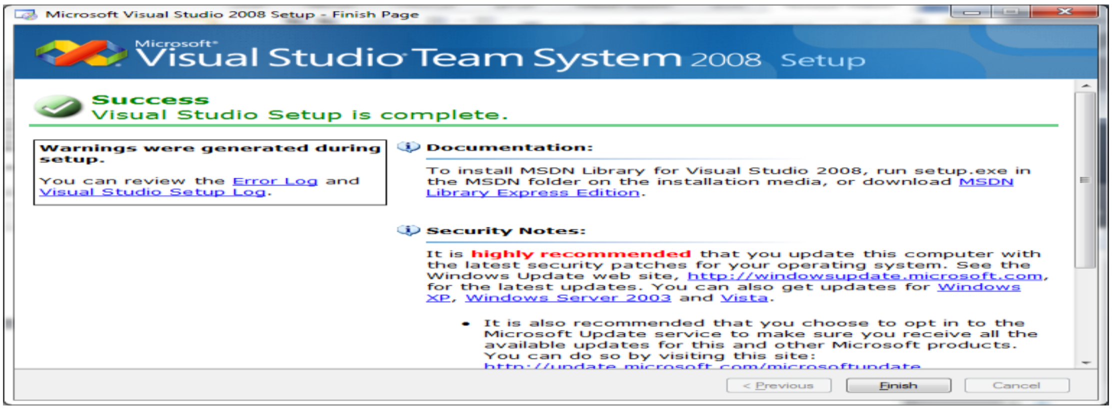
حدد على Default ثم حدد مسار تنصيب الحزمة على جهاز الكمبيوتر ثم إنقر Install .

- تظهر نافذة تنصيب البرامج الواحد تلو الآخر وأي برنامج يتم تنصيبه تظهر أمامه العلامة (/ true(، إنتظر لفترة من الوقت إلى أن يتم تنصيب جميع البرامج .

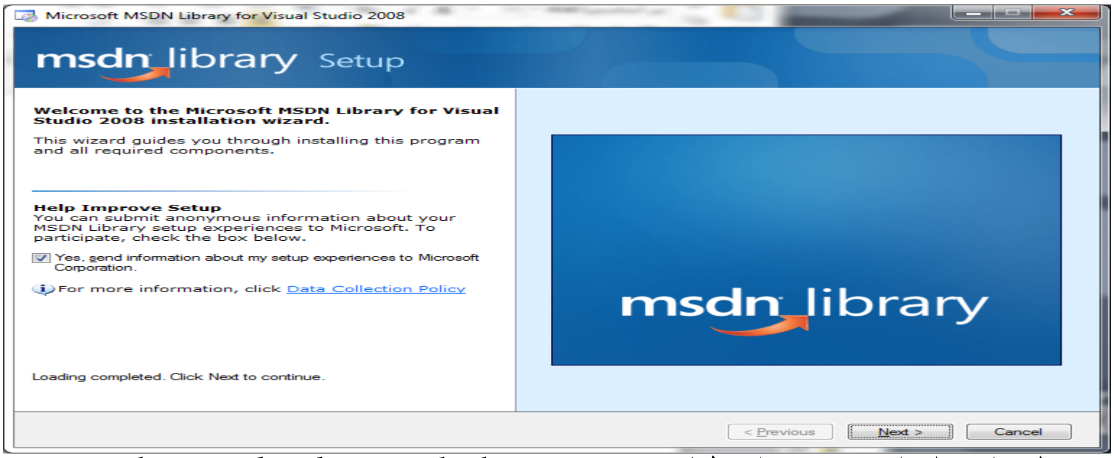


- تظهر نافذة تخبرك بأنه تم تنصيب حزمة Visual Studio 2008 بنجاح

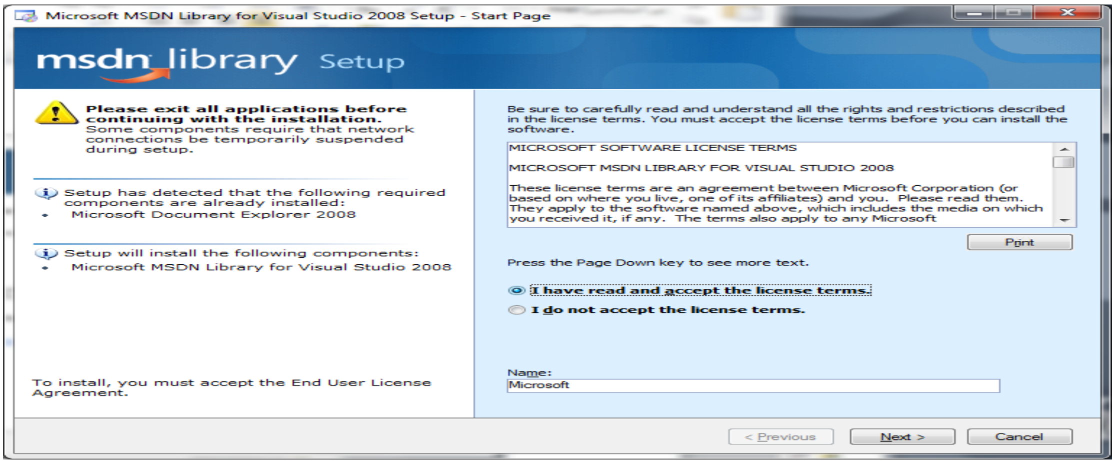
إنقر على الزر Finish .



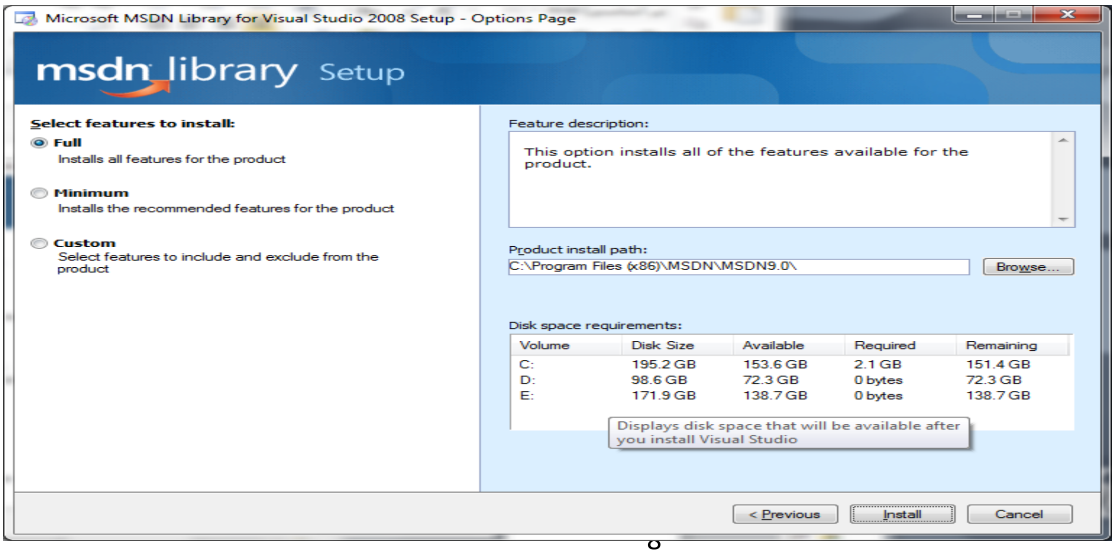
- لتنصيب مكتبة التعليمات MSDN عد إلى جهاز الكمبيوتر -> قرص DVD إنقر عليه بالزر الأيمن واختر قراءة تلقائية تظهر لك نافذة إنقر على الخيار الثاني Install Product Documentation– تظهر نافذة ترحيبية ... إنقر الزر Next



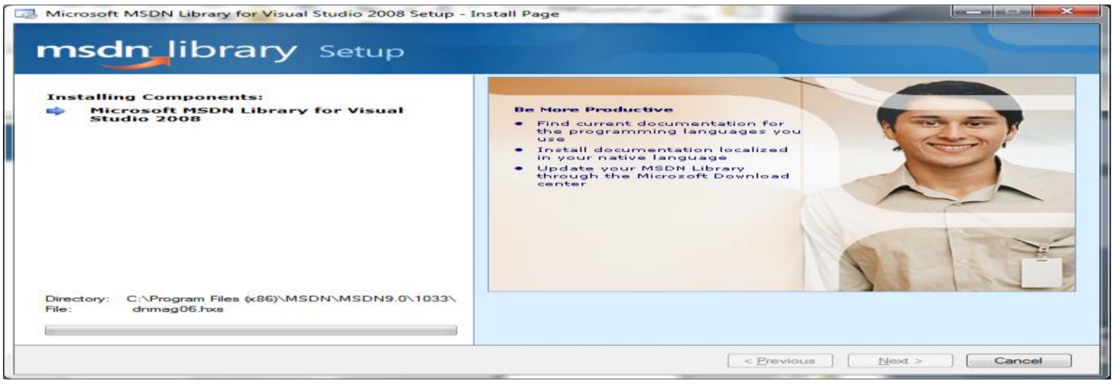
- تظهر نافذة الإتفاقية ... إنقر على الخيار I have read and accept the license terms ثم إنقر على الزر Next



- تظهر نافذة إختر Full ثم حدد المسار وأنقر على الزر Next



- تظهر نافذة Install page لتنصيب MSDN Library كما في الصورة :



... وانتظر حتى يتم تنصيب الـ MSDN كاملة بعدها إنقر الزر Finish .

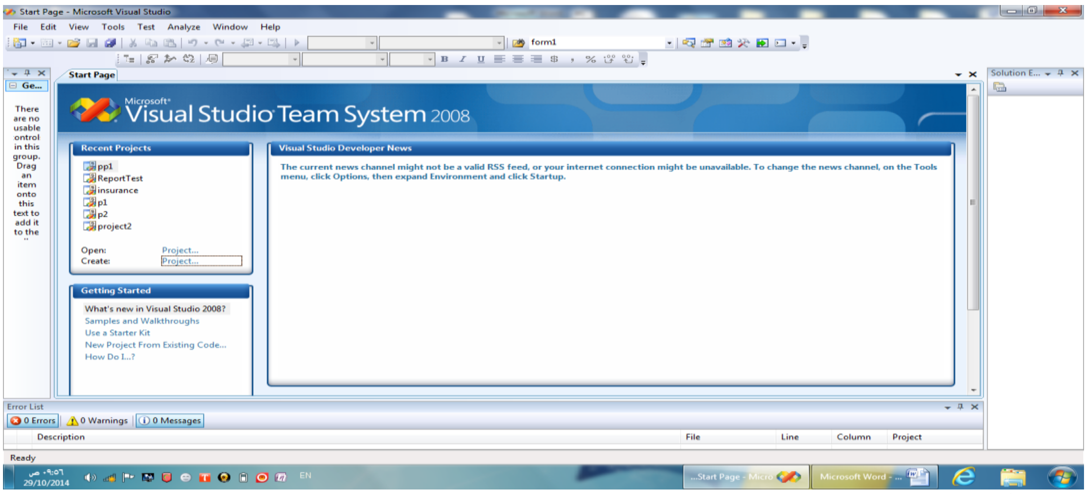
- كيفية فتح حزمة Visual Studio 2008 : توجد عدة طرق لفتح حزمة VS.Net

- من قائمة إبدأ -> كافة البرامج - > مجلد Microsoft Visual Studio 2008 -> أيقونة Microsoft Visual Studio 2008 .

- أو بالإمكان النقر نقراً مزدوجا على الأيقونة إذا كانت موجودة كإختصار للغة على سطح المكتب .

- تظهر نافذة تهيئة الحزمة بأي لغة من لغات البرمجة الموجودة ضمن الحزمة إختر Visual C# Devlopment Setting ثم إنقر الزر OK .

- تظهر نافذة البداية Start Pageكالتالي :



وتتكون من النوافذ التالية :

= Recent Projects : وتضم قائمة بآخر المشاريع التي تم العمل عليها وكذلك الزرين :

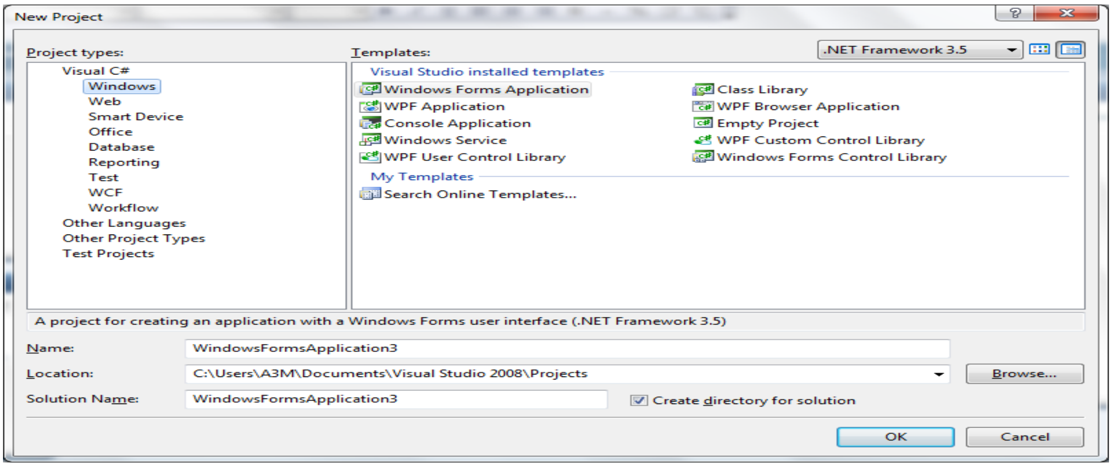
Open : Project (فتح مشروع موجود مسبقاً)

Create: Project (لإنشاء مشروع جديد)

= Getting Started : ويضم مواضيع عن جديد اللغة في هذا الإصدار ومواضيع أخرى .

= Visual Studio Developer News : ويحتوي معلومات حول تطوير اللغة online .

- برنامجك الأول بلغة Visual C#.Net :



وتحتوي على عدة أجزاء كما يلي :

= Project Type : نوع المشروع وبأي لغة برمجة يعمل ، تحت تشجير Visual C# إختر النوع Windows .

= اللسان NET Framework 3.5. وهو الإصدار الأخير للمنصة التي تعمل عليها لغات الـ Visual studio 2008 وتوجد أيضاً الإصدارات السابقة 2.0&3.0 .

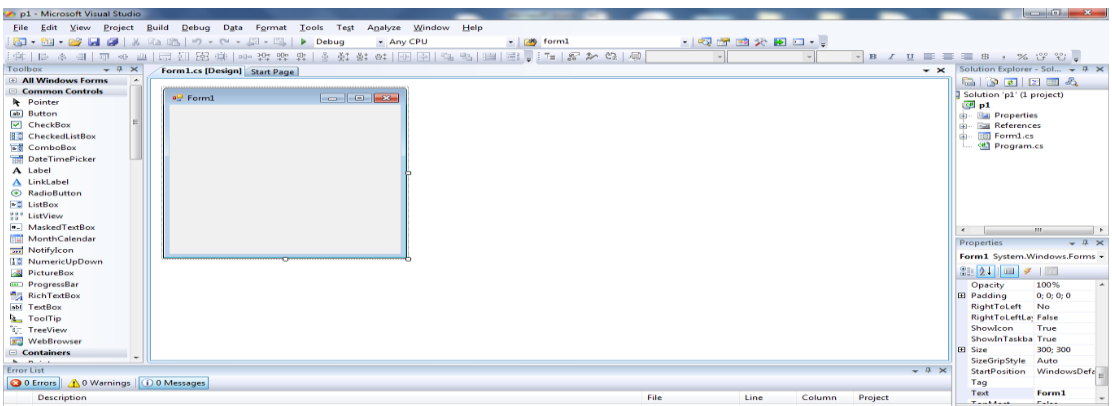
= Template : نوع قالب المشروع ، نختار Windows Forms Application .

= Name : إسم المشروع وغالباً ما يكون الإسم الإفتراضي WindowsFormsApplication1 قم بتغيير إسم المشروع بإسم p1 .

= Location : مكان حفظ المشروع ويمكن تغيير المسار بالنقر على الزر Browser وتحديد مكان حفظ المشروع ، إترك المسار الإفتراضي C:\Users\A3M\Documents\Visual Studio 2008\Projects

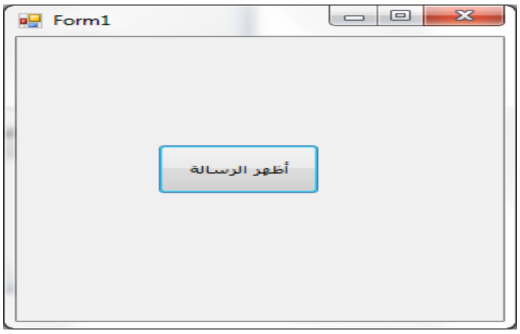
= Solution Name : إسم الحل الذي ينتمي إليه المشروع وغالباً ما يكون بنفس إسم المشروع ويمكن تعديله ، إترك الإسم الإفتراضي كما هو، ثم إنقر على الزر Ok .

- الآن تظهر لنا بيئة التطوير المتكاملة IDE والتي يتم فيها عمل المشروع بجميع المراحل من التصميم المرئي وكتابة الأكواد البرمجية وحتى ترجمة المشروع وتنفيذه .



الآن سنقوم بتنفيذ البرنامج الأول ، عليك إتباع الخطوات بدقة :

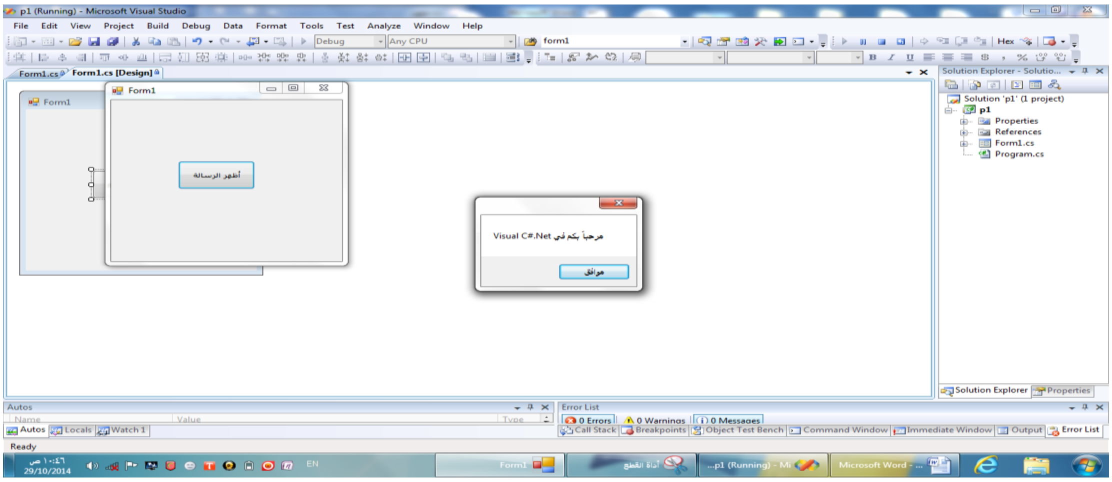
أضف أداة الزر Button من نافذة الأدوات إلى الـ Form1 واسحبه إلى منتصف النموذج ثم إضبط خاصية الـText له بالقيمة "أظهر الرسالة" من خلال نافذة الخصائص بحيث يظهر كالتالي:



- إنقر نقراً مزدوجاً على الـbutton1 للدخول إلى نافذة الـCode يظهر الحدث الإفتراضي Click داخل الحدث إكتب الكود التالي :

```
privatevoid button1_Click(object sender, EventArgs e)
                 {
           MessageBox.Show(" Visual C#.Net مرحباً بكم في ");
                 }
```

- قم بتنفيذ البرنامج بالضغط على الزر ▶ ود في شريط الأدوات أو بالضغط على المفتاح F5 من لوحة المفاتيح ، تظهر النافذة إنقر على الزر "أظهر الرسالة" يظهر الشكل التالي :



- نلاحظ ظهور رسالة بعد النقر على الزر إنقر موافق ثم إخرج من زمن التنفيذ بالضغط على الزر ✖ الـ Form ، يعود البرنامج إلى زمن التصميم .

- للخروج من اللغة إنقر على الزر ✖ أعلى يمين الشاشة أو من القائمة File -Exit< ، يتم الخروج من بيئة الـ Visual studio 2008.

- الملف التنفيذي للبرنامج ومساره :

إذهب إلى مجلد المستندات ->Visual Studio 2008 ->Projects نلاحظ أن وجود مجلد بإسم p1 وهو مجلد مشروع البرنامج الأول الذي قمنا بعمله سابقاً قم بفتح المجلدات كالتالي :

p1 -> bin -> Debug

نلاحظ وجود ٤ ملفات ... يوجد ملف بإسم p1بإمتدادexe يعتبر الملف التنفيذي للبرنامج p1 قم بالنقر عليه نقراً مزدوجاً تظهر نافذة الـ Form1 إنقر على الزر "أظهر الرسالة" تظهر الرسالة مثل التي ظهرت أثناء تنفيذ البرنامج أثناء فتح اللغة ... يعتبر هذا الملف نسخة تنفيذية للمشروع يمكن عمل إختصار لها على سطح المكتب لكي يتم تشغيله والعمل عليه بدون الحاجة إل الدخول لمجلد البرنامج أو فتح حزمة Visual Studio 2008وتنطبق هذه العملية على جميع المشاريع التي يتم إنجازها على بيئة NET. .

الفصل الثاني

مفاهيم برمجية في لغة Visual C#.Net

قبل البدء في عمل المشاريع والبرامج بلغة VC#.Net هناك مفاهيم وقوانين برمجية يجب مراجعتها بدقة لكي لا تحدث أخطاء خلال كتابة الأكواد البرمجية يمكن سرد هذه المفاهيم كالتالي :

- أنواع البيانات :Data Type

النوع	السعة بالبايت	الوصف	أمثلة
Byte	1	بايت	00000001 = 1
Int	4	صحيح	120 , 0 , -9
Long	8	طويل	55 , 0 , -190
Float	4	حقيقي	5f , -22f
Double	8	مضاعف	90 , 2e+10 , 55.8e-10
Decimal	8	عشري	88 , -9006
String	________	سلسلة حرفية (نصي)	"Hello in C#"
Char	1	حرفي (رمزي)	'A' , 'n' , '4'
bool or Boolean	1	منطقي	True or False

- العمليات الحسابية :Arithmetic Operators

العملية	وصف العملية	مثال	النتيجة
-	سالب	-2	-2
*	ضرب	2 * 3	6
/	قسمة	3 / 2	1.5
%	باقي القسمة	17 % 5	2
+	جمع (إضافة)	2 + 3	5
-	طرح	3 – 2	1

- عمليات المقارنة :Comparison Operators

العملية	الوصف	مثال	النتيجة
==	يساوي	7 == 2	False
>	أكبر من	6 > 3	True
<	أصغر من	5 < 11	True
>=	أكبر من أو يساوي	23 >= 23	True
<=	أصغر من أو يساوي	4 <= 21	True
!=	لا يساوي	3 != 3	False

- العمليات المنطقية :Logical Operators

النتيجة	مثال	الوصف	العملية
True	(2<3) && (4<5)	AND	&&
True	(2<3) || (6<7)	OR	||
False	!(3=3)	NOT	!

- شروط تسمية المتغيرات :Variable Name Condition

١- أن يبدأ بحرف أبجدي أو شرطة تحتية ولا يبدأ برقم أو رمز خاص .

٢- يمكن أن يتخلله رقم أو رمز خاص .

٣- يفضل أن يكون إسم المتغير دال على وظيفته .

٤- يفضل أن لا يكون إسم المتغير كبيراً .

٥- يتحسس لحالة الأحرف فإذا كان Capital فلا يجوز كتابته لاحقاً في البرنامج Small

٦- أن لا يكون كلمة محجوزة أو أمر من أوامر اللغة أو إسم Class أو Namespace ...

- أمثلة لمتغيرات تسميتها مقبولة :

number1 , A$, _result , A3M ...

- أمثلة لمتغيرات تسميتها غير مقبولة :

#q1 , 67X , this , Textbox ,using ...

- محددات الوصول : Accessibility وتكتب قبل جملة تعريف المتغير وهي كالتالي :

- public : (عام) يمكن رؤيته في جميع أجزاء البرنامج حتى من Form أو Class آخر .
- private : (خاص) لا يمكن رؤيته إلا في نفس الـForm أو الـ Class .
- static : (ساكن) ويمكن رؤيته خارج الأحداث على الـ Form ويحتفظ بآخر قيمة .
- default : (إفتراضي) وفي هذه لا يكتب أي محدد وصول أمام المتغير .
- local : (محلي) بحيث يتم تعريف المتغير داخل الحدث لا يمكن إستخدامه في حدث آخر أو يتم تعريفه داخل {}block لا يمكن إستخدامه خارج الـ block .

مثال : public int x ;

- الثوابت : Constants وتكون قيمة المتغير ثابتة لا يمكن تغييرها

مثال : const double pi=3.14 ;

- جمل الشرط : Condition Statements

- الشرطية الإستفهامية ؟ : وتكتب في سطر واحد مثلاً برنامج القيمة المطلقة للعدد :

```
int x = -9, y;
y = (x > 0) ? x : -x;
MessageBox.Show(y.ToString());
```

- if الشرطية : ولها عدة صيغ Syntax كالتالي :

١- if المفردة :

```
if (condition) Statement ;
or
if (condition) {
statements;
..........
}
```

٢- if ... else :

```
if (condition)
Statement1;
else
Statement2;
```

٣- if ... else if ... else :

```
if ( condition1)
Statement1;
else if (condition2)
Statement2;
........
else
Statementn;
```

- مثال على if الشرطية :برنامج يميز الأعداد الزوجية والأعداد الفردية

```
int x = 11;
if (x % 2 == 0)
MessageBox.Show(x.ToString() + "عدد زوجي");
else
MessageBox.Show(x.ToString() + "عدد فردي");
```

- جملة switch case الشرطية : وتكون صيغتها بالشكل التالي :

switch (VarName) {

case Value1 : Statement1 ; break ;

case Value2 : Statement2 ; break ;

………………

default: Statementn ; break ;

}

- مثال على جملة switch : برنامج لعرض إسم العدد المدخل

```
int x = 1;
switch (x)
            {
case 1: MessageBox.Show("one"); break;
case 2: MessageBox.Show("two"); break;
case 3: MessageBox.Show("three"); break;
default: MessageBox.Show("not found"); break;
            }
```

- الحلقات التكرارية : Looping Statements

- حلقة for :وتكون صيغتها بالشكلالتالي :

for (initExpression ; condition ; incrementExpression)

{

statements ;   }

- مثال على الحلقة for :برنامج لعرض الأعداد من 1 إلى 10

```
for (int x = 1; x <= 10; x++)
            {
MessageBox.Show(x.ToString());
            }
```

- مثال على حلقة for المتداخلة : برنامج لعرض جدول الضرب للأعداد من 1 إلى 12

```
for(int i = 1; i <= 12 ;i++)
for (int j = 1; j <= 12; j++)
               {
MessageBox.Show((i * j).ToString());
}
```

- حلقة while : وتكون صيغتها بالشكل التالي :

initExpression ;

while(condition)

{

statements;

incrementExpression;}

- مثال على حلقة while : برنامج لعرض مربعات الأعداد من 1 إلى 10

```
int i = 0;
while (i < 10)
            {
                i++;
MessageBox.Show((i * i).ToString());
            }
```

- حلقة do – while :وتكون صيغتها بالشكل التالي :

initExpression ;

do {

statements;

incrementExpression;

} while(condition) ;

- مثال على حلقة do – while : برنامج لعرض مكعبات الأعداد من 1 إلى 10

```
int x = 0;
do
            {x++;
MessageBox.Show((x * x * x).ToString());
          } while (x < 10);
```

- حلقة foreach :

سنوضحها بالمثال التالي : برنامج لعرض عناصر مصفوفة نصية (أسماء الدول)

```
string [] x ={"اليمن","مصر","الأردن"} ;
foreach(string sin x)
MessageBox.Show(s);
```

// ملاحظة : لتنفيذ جميع الأكواد السابقة واللاحقة في VC#.Net يجب كتابتها داخل حدث معين مثلاً حدث الـClick button وذلك بعد إضافة الزر إلى الـ Form .

- المصفوفات : Arrays

هي عبارة عن كتلة من الذاكرة كبيرة بما يكفي لتحمل عدة متحولات من نفس النوع وكل عنصر لديه دليل أو فهرس وتوجد مصفوفات أحادية البعد ومصفوفات متعددة الأبعاد .

- مصفوفة أحادية البعد : ويتم تعريفها و إسناد القيم إليها كالتالي :

```
int[] a = newint[10];
for (int i = 0; i < 10; i++)
                a[i] = i + 1;
```

ويمكن إختصار الكود السابق في سطر واحد كالتالي :

```
int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

ولعرض عناصر المصفوفة نكتب الكود كالتالي :

```
for (int i = 0; i < 10; i++)
MessageBox.Show(a[i].ToString());
```

- مصفوفة متعددة الأبعاد : مثلاً ثنائية البعد يتم تعريفها كالتالي :

```
int [ , ] a=newint [3,3];
```

ولإسناد القيم إليها نكتب الكود كالتالي :

```
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
 a[i, j] = i + j;
```

ويمكن تعريف المصفوفة ثنائية البعد وإسناد القيم إليها في سطر واحد كالتالي :

```
int [ , ] a={{1,2,3},{4,5,6},{7,8,9}};
```

ولعرض عناصر المصفوفة ثنائية البعد نكتب الكود كالتالي :

```
for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++)
MessageBox.Show(a[i,j].ToString());
```

- تجمعات البيانات في NET. : وهي تجمعات بيانات جاهزة توفرها بيئة الـ NET. يتم إنشاء كائن منها لإستخدامها عبر طرق وخصائص خاصة بها ولإستخدامها يجب كتابة هذه الجملة في قسم التصريحات أعلى البرنامج : `using System.Collections;` وهي كالتالي :

- ArrayList :وهي مصفوفة ليس لها حدود ويتم إنشاء كائن منها كما يلي :

```
ArrayList list = newArrayList();
```

- Stack : يعمل عمل المكدس بخوارزمية LIFO ويتم إنشاء كائن منه كالتالي :

```
Stack s = newStack();
```

- Queue : يعمل عمل الطابور بخوارزمية FIFO ويتم إنشاء كائن منه كالتالي :

```
Queue q = newQueue();
```

- Hashtable : جدول التجزئة وهو بنية بيانات تستخدم كائناً كمفتاح فريد ويتم إنشاء كائن منه

```
Hashtable mybills = newHashtable();
```

- بعض أصناف التجمعات العامة :

- List : ويستخدم في الحالات التي تستخدم فيها ArrayList ويشتق منه كائن كالتالي :

```
List<int> myInts = newList<int>();
```

وتوجد أصناف أخرى مثل Queue,Stack,SortedList,Dictionary ...

- مفاهيم في البرمجة كائنية التوجه OOP في لغة C# :

- Class صنف  يتكون من دوال ودوال بناء ومتغيرات ويمكن أن يرث Class آخر ويمكن أن ينفذ Interface وتوجد Classes جاهزة في لغة VC# حتى أن كل أداة تعتبر Class وأيضاً

تشتق من Class أب أعلى منها فمثلاً أداة مربع النص TextBox وأداة مربع النص الغني RichTextBox كلاهما تشتق من الصنف TextBoxBase ...

- Interface واجهة :وتمتلك دوال مجردة ومتغيرات ثابتة بحيث يتم إعادة تعريف الدوال المجردة في الـClass الذي ينفذ الـInterface وكذلك كتابة أكواد فيها ويمكن للـInterface أن ترث Interface أخرى وتوجد في VC# عدة Interfaces جاهزة مثل `IQueryProvider` ...
- Object كائن : ويتم تعريف الـObject من نوع الـClassوإطلاق دالة بناء منه لكي يتم إستخدام طرق وخصائص وعناصر ذلك الـClass فمثلاً لإنشاء Object من الصنف Form1 نكتب الكود التالي :

```
Form1 frm = newForm1();
```

- فضاء الأسماء NameSpace {} : ويحتوي على عدة Classes وتوجد عدد من الـnamespace الجاهزة في VC# ويجب إستيرادها بجملة Usingلإستخدام الـClasses التي تحتويها في بداية البرنامج أعلى نافذة الـCode فمثلاً فضاء الأسماء الذي تشتق منه الأدوات Controls هو `using System.Windows.Forms;`
- المعددات Enum : ويحتوي أكثر من قيمة في نفس الوقت وتوجد عدة معددات جاهزة في لغة VC# مثلاً `FormWindowState.Minimized-FormWindowState.Maximized - FormWindowState.Normal` ...

وتوجد مفاهيم أخرى مثل struct بنيه –property خاصية ...

// ملاحظة :كل المسميات التي تم ذكرها يمكن بناؤها من قبل المبرمج وإستخدامها داخل لغة VC# وفق القواعد المحددة لكل مفهوم .

- مفاهيم في البرمجة المرئية بلغة VC# :

- Solution حل وهو عبارة عن حل لمشكلة برمجية ويمكن أن يحتوي على Project واحد أو أكثر ويمكن أن يكون كل Project بلغة أخرى أو من نوع آخر ضمن بيئة Net. .
- Project مشروع بر الـProject برمجياً namespace ويحتوي كل Project على عدة ملفات ضمن ٣ تفرعات شجرية :

١- مجلد References ويضم عدة مكتبات رئيسية التي تستخدم ملفاتها في بناء المشروع .

٢- مجلد Properties (خصائص المشروع) ويحتوي عدة ملفات تمثل إعدادات المشروع .

٣- النموذج Form الذي يتم تصميم البرنامج عليه ويضم عدة ملفات تحتوي أكواد برمجية .

ويمكن إضافة أكثر من Form إلى الـProject واحدوكذلك إضافة ملفات مثل قواعد البيانات وملفات صور ....

- Tool أداه:ي أيقونة موجودة في صندوق الأدوات Toolbox لكل أداة وظيفة معينة يتم إضافتها إلى سطح الـForm يدوياً في زمن التصميم Design أو برمجياً في زمن الـCode وتمتلك كل أداه خصائص وأحداث وطرق ويمكن ضبط خصائص الأداة يدوياً من نافذة الخصائص في زمن التصميم أو برمجياً في زمن كتابة الـCode أما الأحداث والطرق فيتم إستخدامها في زمن كتابة الـCode ويتم إستخدام الأدوات من قبل المستخدم في زمن التنفيذ .
- property خاصية : هي التأثير الذي يحصل على الأداة في حال تغيير قيمتها ولإظهار خصائص الأداة يتم التحديد على الأداة والتوجه نحو نافذة الـخصائصproperties أو تظهر في نافذة الـCode بعد كتابة نقطة (.) بعد الإسم البرمجي للأداة .
- event حدث : هو الفعل الذي يقوم به المستخدم على الأداة في زمن التنفيذ بحيث يتم الإستجابة بتنفيذ الكود البرمجي الموجود داخل الحدث والذي يكون محصوراً داخل {} .
- method طريقة : هي وظيفة معينة تتبع كل أداة بحيث تحدث فعلاً على تلك الأداة بناءً على نوع وظيفة هذه الطريقة ودائما ما تتبع بـ () ويمكن أن يحتوي القوسين على بارامتر أو أكثر حسب صيغة الطريقة نفسها وكل أداة تمتلك طرق جاهزة ويمكن إنشاء طريقة داخل نافذة الـCode وإستخدامها داخل الأحداث وفي هذه الحالة تكون تابعة للـForm .

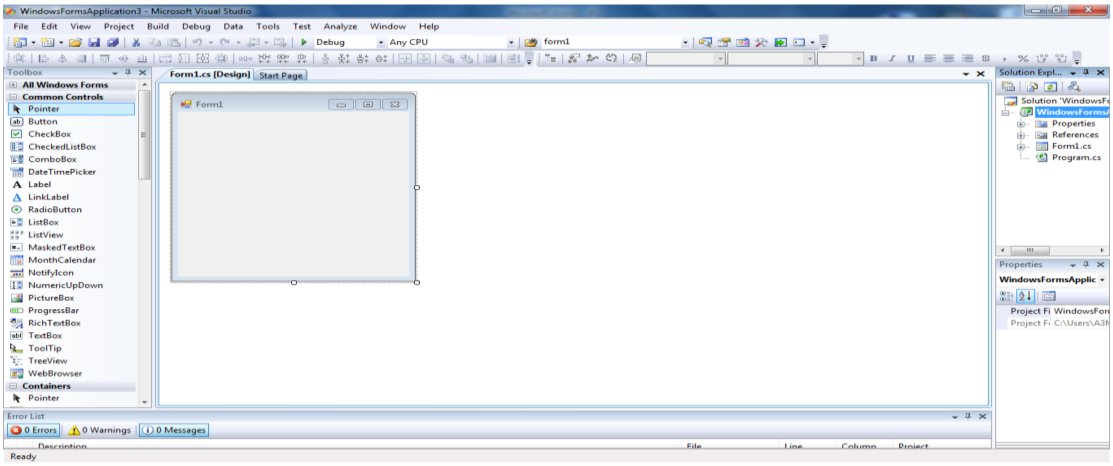
//كل أسماء الأدوات والخصائص والأحداث والطرق ... تعتبر كلمات محجوزة .

الفصل الثالث

بيئة التطوير المتكاملة IDE

Integrated Development Environment

- بيئة التطوير المتكاملة IDE : هي بيئة تطوير برمجيات تساعد في تصميم وكتابة وترجمة وتنقيح وحتى حزم براجك وتتكون من مجموعة من الأشرطة و القوائم والنوافذ...تساعد المبرمج على إنجاز تطبيقاته بكل سهولة ويسر وبشكل مرئي ...



- مكونات بيئة التطوير المتكاملة :

- شريط العناوين Title Bar : 

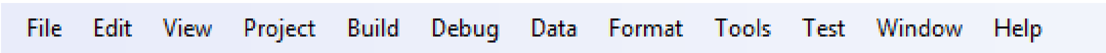
ويتضمن العناصر التالية :-

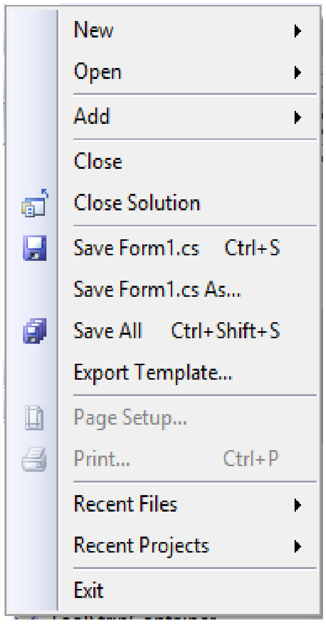
إسم المشروع الحالي – جملة "Microsoft Visual Studio " & أيقونة حزمة VS2008 .

صندوق التحكم Control Box : ويحتوي على الأزرار التالية :

: زر إغلاق نافذة VS - : زر تحجيم النافذة زر تصغير النافذة .

- شريط القوائم :Menus Barويتضمن عدة قوائم بحيث تحتوي كل قائمة مجموعة من الأوامر وكل أمر يؤدي وظيفة معينة .. كما في الصورة :

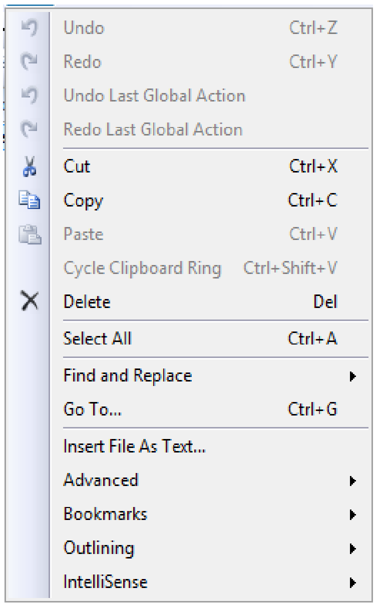




١- File (ملف) : وتحتوي على الأوامر التالية كما في الصورة :

- New<ـــProject : لإنشاء مشروع جديد .
- Open<ـــProject\Solution : لفتح مشروع موجود مسبقاً .
- Add<ـــNew Project : لإضافة مشروع جديد إلى الـ Solution الحالي .
- Close : إغلاق نوافذ الـProject مع بقاء نافذة Solution وتظهر Start Page .
- Close Solution : إغلاق كافة نوافذ الـSolution وتظهر Start Page .
- Save Form1.cs : لحفظ التعديلات على الـForm الحالي بدون تغيير الإسم .
- Save Form1.cs As.. : لحفظ الـForm الحالي بإسم جديد .

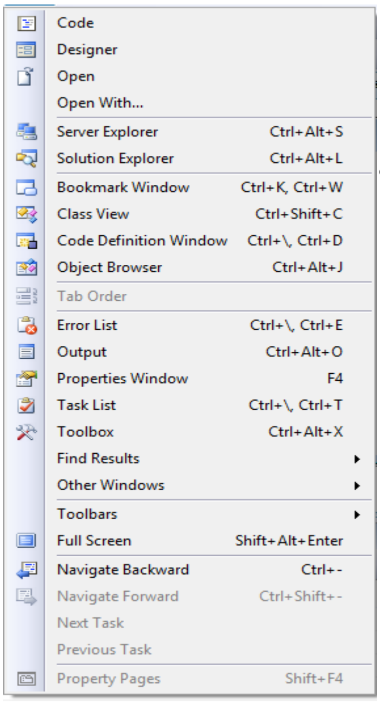
- Save All : لحفظ التعديلات على جميع نوافذ الـProject .
- Export Template : لتصدير قالب من الـProject الحالي بعد حفظ جميع عناصر المشروع.
- Page Setup : يفعل على شاشة الـCode لإعداد الصفحة للطباعة .
- Print : لطباعة الـCode الحالي .. حدد الطابعة وعدد النسخ ثم إنقر OK .
- Recent Files : لعرض قائمة بآخر ملفات تم فتحها داخل بيئة التطوير المتكاملة IDE .
- Recent Projects : لعرض آخر المشاريع التي تم فتحها داخل الـIDE .
- Exit : للخروج من Visual Studio بشكل كامل .



٢- Edit(تحرير) : وتحتوي على الأوامر التالية كما في الصورة :
(هذه الأوامرتفعل غالبا على شاشة الـCode)

- Undo : تراجع للأمام . –Redo : تراجع للخلف .
- Cut : قص . –Copy : نسخ . –Past : لصق .
- Delete : حذف . –Select All : تظليل الكل .
- Find and Replace : بحث وإستبدال ويحتوي عدة خيارات منها :
Quick Find : للبحث عن كلمة داخل الـCode .
Quick Replace : لإستبدال كلمة بدلاً عن أخرى داخل الـCode .

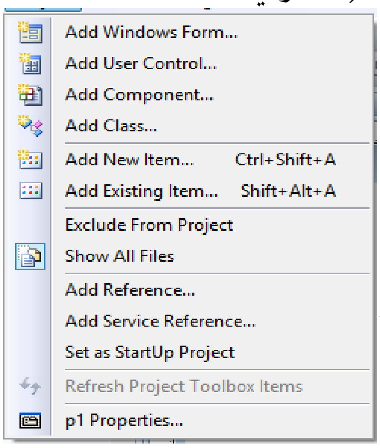
٣- View (عرض) : للتحكم في ظهور نوافذ بيئة التطوير المتكاملة IDE كما في الصورة :



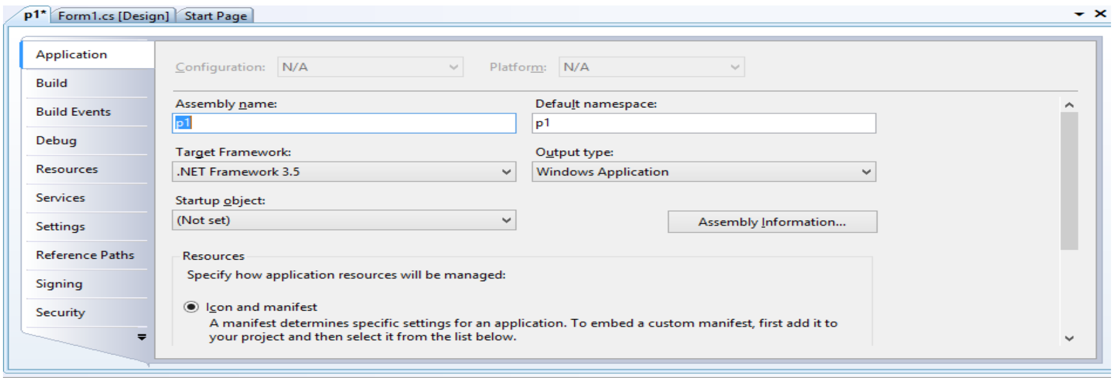
- Code : نافذة كتابة الأكواد البرمجية .
- Designer : نافذة التصميم .
- Server Explorer : نافذة مستكشف السيرفرات وتستخدم في قواعد البيانات
- Solution Explorer : نافذة مستكشف الحل ويحتوي على Project أو أكثر
- Class View : نافذة الأصناف وفضاءات الأسماء الخاصة بالبرنامج الحالي .
- Object Browser : نافذة مستعرض الكائنات الموجودة في لغة VC#.NET .
- Error List : نافذة قائمة الأخطاء الموجودة في أكواد البرنامج الحالي .
- Properties Window : نافذة الخصائص التي تخص جميع الأدوات والـForm
- Toolbox : نافذة صندوق الأدوات .. يتم إضافة الأدوات يدوياً منه .
- Other Windows : تعني قائمة بنوافذ أخرى مثل :
Start Page : صحة البداية ...
- Toolbar : وتضم قائمة بأشرطة الأدوات مثل Build : شريط أزرار بناء وترجمة الـSolution

- Full Screen : لعرض بيئة التطوير المتكاملة ملء الشاشة .

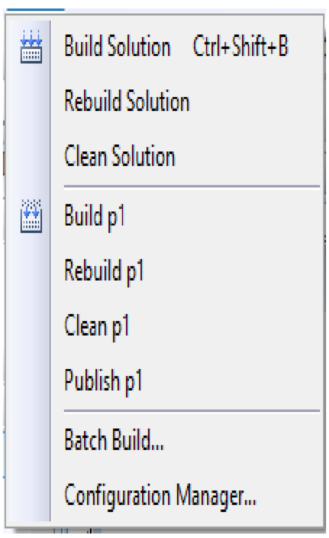
٤- Project (مشروع) : تحتوي على أوامر لإضافة عن عناصر جديدة إلى الـProject وهي :



- Add Windows Form : لإضافة نموذج جديد إلى المشروع .
- Add User Control : لإضافة نافذة تحكم مستخدم إلى المشروع .
- Add Component : لإضافة مكون جديد إلى المشروع .
- Add Class : لإضافة صنف جديد إلى المشروع .
- Add New Item : لإضافة عنصر جديد للمشروع .. تظهر نافذة حدد العنصر المراد إضافته ثم أكتب إسمه البرمجي ثم إنقر على الزر Add .
- Add Existing Item : لإضافة عنصر موجود في مشروع سابق .
- Show All Files : لإظهار كافة عناصر المشروع في نافذة الـSolution Explorer .. تظهر مجلدات تحتوي ملفات أسفل الـProject .
- Add Reference : لإضافة مكتبات جديدة إلى المـشروع .
- P1 Properties : لإظهار نافذة خصائص المشروع .. توجد فيها عدة تبويبات يحتوي كل تبويب على مجموعة من الخصائص كما في الصورة :



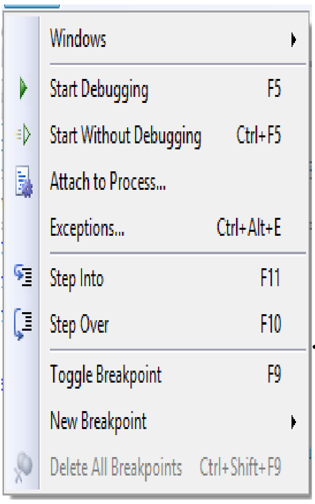
٥- Build (بناء – ترجمة): يحتوي على أوامر لترجمة الـProjects الموجودة داخل Solution كما في الصورة :



- Build Solution : ترجمة جميع الـProjects الموجودة داخل الـSolution الحالي .. إنتظر قليلاً حتى يتم بناء الحل وفي حال نجاح العملية تظهر رسالة في شريط الحالة Build Successful وفي حال وجود أخطاء برمجية تفشل العملية و تظهر رسالة Build Failed كما تظهر الأخطاء في نافذة Error List .
- Rebuild Solution : إعادة بناء وترجمة الـSolution في حال تم ترجمته مسبقاً وحدثت تعديلات جديدة على ملفات الـProjects الموجودة داخله .

- Clean Solution : مسح وإزالة عملية بناء وترجمة الـSolution السابقة .
- Publish p1 : لنشر الـProject الحالي على جهاز الكمبيوتر أو على Server أو على موقع أنترنت بحيث تظهر أيقونة بصيغة تنفيذية يعمل خلالها البرنامج في زمن التنفيذ قط .
- Configuration Manager : لإدارة تشكيل المشاريع ويستخدم هذا الخيار غالباً أثناء حزم البرنامج كملف تنفيذي Setup ... سيتم شرحها لاحقاً .

٦- Debug (تنفيذ) : يحتوي على أوامر تخص الانتقال بالبرنامج إلى طور التنفيذ كما في الصورة :



- Windows : يحتوي على قائمة بالنوافذ التي تخص زمن التنفيذ وهي :

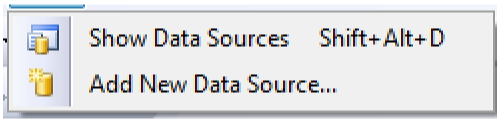
Breakpoints : نافذة لإنشاء أو حذف نقاط الإنقطاع في نافذة الـCode

Output : نافذة المخرجات ويظهر فيها معلومات عن thread ...

Immediate : نافذة الخرج الفورية .

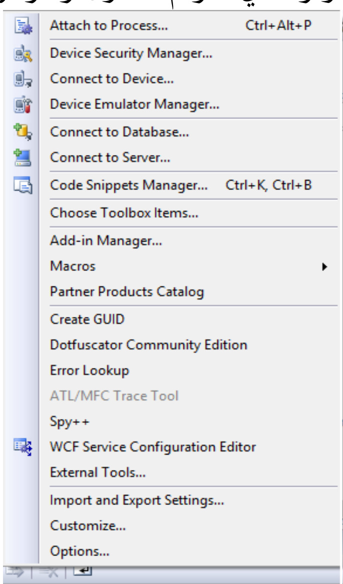
- Start Debugging : لبدء تنفيذ البرنامج . - Stop Debugging : إيقاف التنفيذ.
- Start Without Debugging : لتنفيذ البرنامج من خارج الـDebug .
- Exceptions : نافذة الإستثناءات التي قد تظهر في البرنامج .

٧- Data : تحتوي على أوامر تتعامل مع مصادر البيانات .. خاصة قواعد البيانات كما في الصورة :

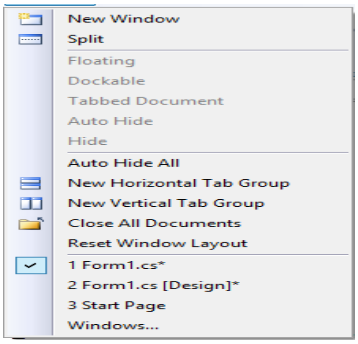


- Show Data Sources : لإظهار نافذة مصادر البيانات الموجودة ضمن البرنامج .
- Add New Data Sources : لإضافة مصدر بيانات جديد .

٨- Tools : تحتوي على أدوات ذات خصائص متنوعة التي لا توافق بشكل خاصعم القوائم الأخرى وتحتوي أيضاً على قليل من الأوامر المزدوجة في القوائم الأخرى والأوامر التي تعدل في بيئة التطوير المتكاملة IDE نفسها .

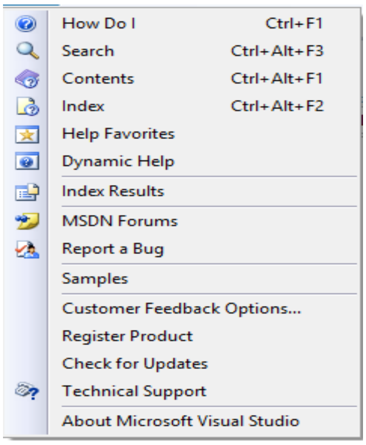


-٩ Window : تحتوي على الأوامر التي تتحكم بنافذة حزمة Visual Studio، هذه الأوامر تفعل إعتماداً على نوع النافذة التي تمتلك التركيز .

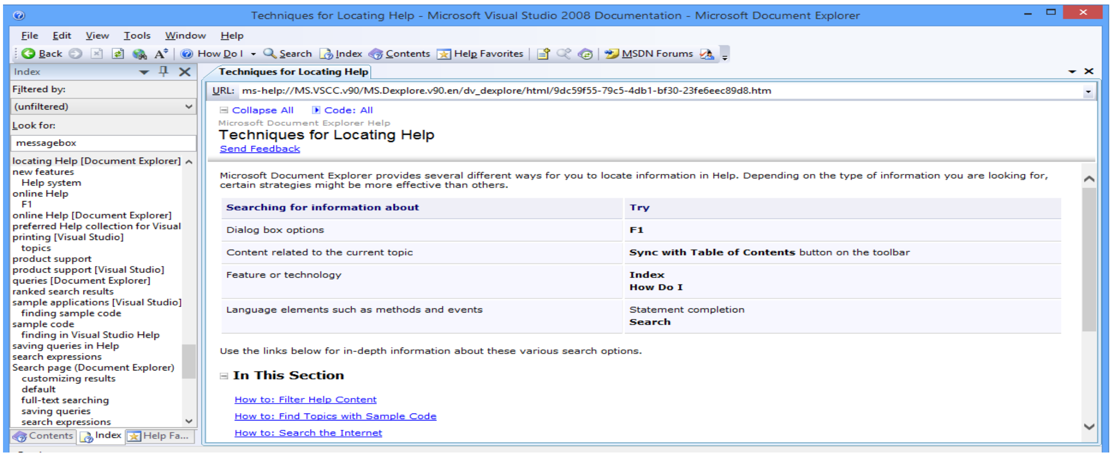


-١٠ Help (مساعدة) : هذه القائمة تعرض عادة مجموعة منوعة من أوامرالمساعدة ، يجب أن تكون مألوفاً مع معظم هذه الأوامر من خبراتك السابقة .

أهم أوامر هذه القائمة كالتالي :-

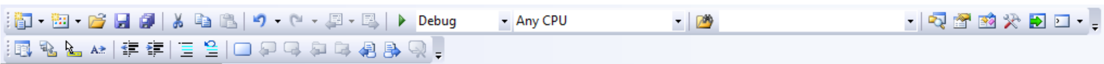


- Index : فهرس بكل محتويات حزمة Visual Studio سواءً كانت Class أو Property أو Event أو Method أو Namespace ... كما توفر Syntax تراكيب لكل ما سبق وكذلك Examples أمثلة باللغات الأربع الموجودة في حزمة Visual Studio ويكون العمل أفضل في الـHelp إذا تم تنصيب مكتبة MSDN .. لاحظ إمكانيات نافذة الـIndex في الصورة :



// ملاحظة : أغلب الأوامر الموجودة في القوائم لديها إختصارات من لوحة المفاتيح مكتوبة أمام كل أمر ...

- شريط الأدوات Tools Bar : يحتوي على أيقونات إختصار للأوامر من قوائم مختلفة يتم الوصول إليها بسهولة وبسرعة أكثر من الأوامر الموجودة فيالقوائم .. لاحظ في الصورة أدناه :



سنقوم بشرح أهم هذه الأيقونات كالتالي :

 : لإنشاء مشروع جديد .  : لإضافة عنصر جديد إلى المشروع Form مثلاً .

: لفتح مشروع موجود مسبقاً . لحفظ التغييرات للنافذة النشطة حالياً من المشروع .

: لحفظ جميع نوافذ وملفات المشروع . : قص . : نسخ . : لصق .

: تراجع إلى الأمام . : تراجع إلى الخلف . للتنقل بين نوافذ المشروع المفتوحة إلى الأمام .

للتنقل بين نوافذ المشروع المفتوحة إلى الخلف . : زر تنفيذ المشروع .

Debug تتبع Solution Configuration . Any CPU

: كسر الكل في زمن التنفيذ . إيقاف أو الخروج من وضع التنفيذ . إعادة تشغيل وضع التنفيذ

: إظهار نافذة Solution Explorer . : إظهار نافذة الخصائص Properties .

: إظهار نافذة Object Browser مستعرض الكائنات . : إظهار صندوق الأدوات Toolbox

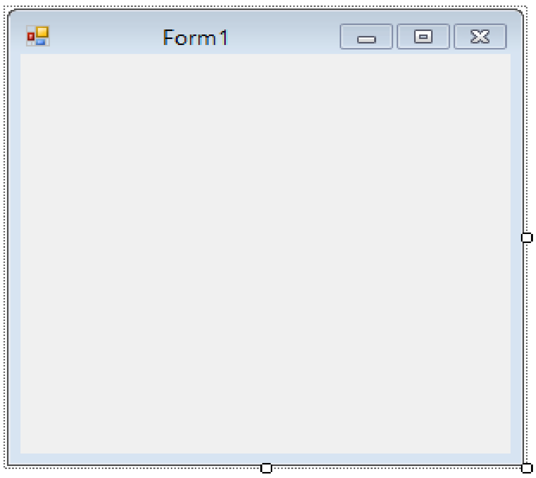
: إظهار نافذة Start Page صفحة البداية . : إظهار نافذة Command Window ...

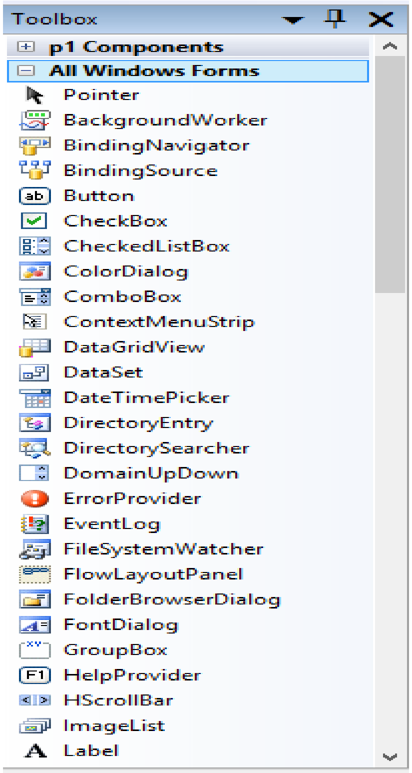
- شريط الحالة State Bar : وهو الشريط الموجود أسفل شاشة Visual Studio وتظهر عليه معلومات عن العمليات التي تحدث على المشروع ... ويسمى أحياناً شريط المعلومات ..

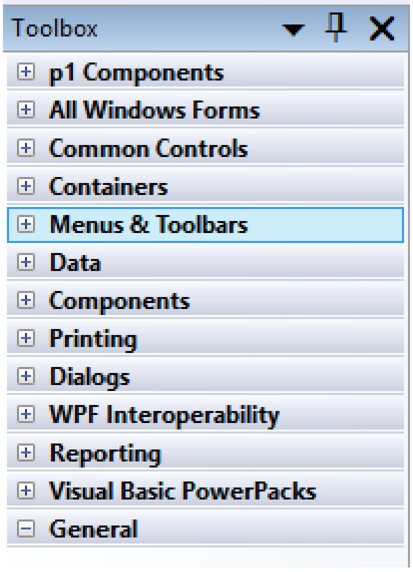


- النوافذ في Visual Studio : تحتوي بيئة التطوير المتكاملة IDE على مجموعة من النوافذ بحيث تحتوي كل نافذة على عناصر أو معلومات وكل نافذة تؤدي وظيفة معينة .. سنتطرق لشرح أهم النوافذ كالتالي :

- نافذة الـ Form Designer : وهي النافذة التي يتم إضافة الأدوات إليها وتتم عليها مرحلة التصميم المرئي وتقع وسط يسار شاشة Visual Studio ... وهي تعتبر ملف داخل المشروع Form1.cs .. لاحظ الصورة :







- نافذة صندوق الأدوات Toolbox : وهي نافذة تحتوي على جميع الأدوات التي يتم إضافتها إلى الـForm يدوياً إما عن طريق النقر المزدوج على الأداة أو بسحب الأداة وإفلاتها على الـForm ويمكن إنشاء الاداة برمجياً .

تم تصنيف الأدوات في الـToolbox إلى مجموعات حسب تقارب شكل ووظيفة الأدوات كالتالي :

All Windows Forms : وتضم معظم الأدوات وأكثرها إستخداماً من مختلف أنواع الأدوات تم ترتيبها أبجدياً حسب إسم الأداة .

Common Controls : وتضم أكثر الأدوات المسماة بـ عناصر التحكم شيوعاً والمشتقة من الصنف Control وتضاف هذه الأدوات على سطح الـForm دائماً أو ما يسمى بـ Client Area .

Containers : وتضم الأدوات التي تستعمل كحاويات بحيث يتم وضع أدوات الـControlغالباً عليها حسب متطلبات البرنامج .

Menus & Toolbars : وتضم الأدوات التي تمثل قوائم وأشرطة والتي تنتهي غالباً بكلمة strip .

Data : وتضم الأدوات التي تستعمل للتعامل مع قواعد البيانات ومصادر البيانات الأخرى .

Component : وتضم عدة أدوات تسمى بالمكونات وهذا النوع من الأدوات لا يضاف على سطح الـForm وإنما أسفل الـForm وتشتق من الصنف Components .

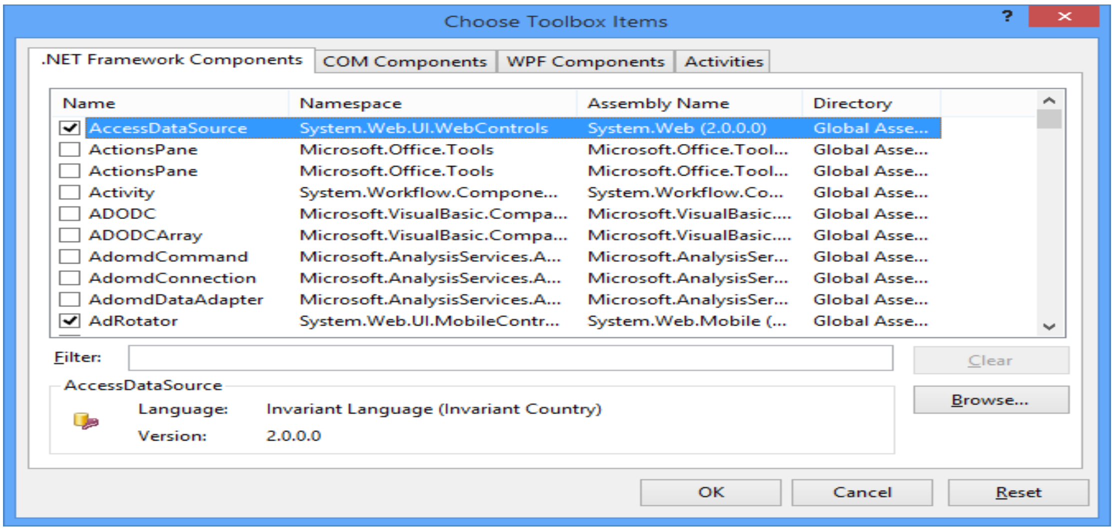
Printing : وتضم أدوات تستخدم في عملية الطباعة ...

Dialogs : وتضم أدوات تسمى بالحواريات مثل تلك الموجودة في أنظمة التشغيل Windows .

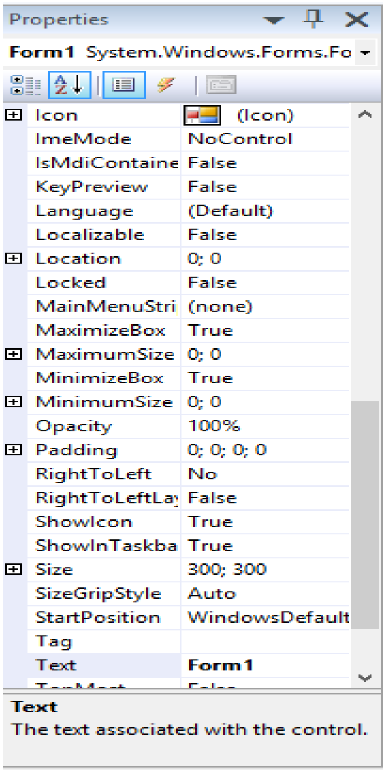
Reporting : وتضم الأدوات المستخدمة في عرض التقارير بأنواعها مثل أدوات الـ Crystal Report وغيرها .

// توجد أدوات أخرى يمكن إضافتها إلى الـToolbox : إنقر بالزر الأيمن داخل Toolbox تظهر قائمة إختر Choose Item تظهر نافذة فيها ٤ تبويبات .. كل تبويب يضم عدد كبير من

الأدوات وأمام كل أداة مربع إختيار.. حدد على الأداة المراد إضافتها ثم أنقر Ok .. بعدها تظهر الأداة المضافة على الـToolbox .. لاحظ صورة نافذة Choose Toolbox Items كالتالي :



// بعض الأدوات غير موجودة ضمن هذه النافذة يتم تنزيلها عبر مواقع الأنترنت أو من مصادر أخرى .. ضع الاداة في أي مكان على جهاز الكمبيوتر ثم أظهر النافذة السابقة .. إنقر على الزر Browse... تظهر نافذة .. حدد على الملف الخاص بالأداة ثم إنقر Open .. سيتم إضافة الأداة إلى التبويب المحدد .. حدد على مربع الخيار أمام الأداة ثم إنقر Ok .. يضاف إلى الـToolbox .



- نافذة الخصائص Properties : وتظهر هذه النافذة بشكل فعال بعد التحديد على الأداة أو الـForm وتضم هذه النافذة خصائص الأدوات وأحداثها .. بحيث تظهر الخصائص إفتراضياً أو بالنقر على الأيقونة أما تظهر الأحداث بعد النقر على الأيقونة ويتم ترتيب عرض الخصائص والأحداث بطريقتين : أبجدياً حسب إسم الأداة أو الحدث بالنقر على الأيقونة أو التصنيف كمجموعات بالنقر على الأيقونة

= بالنسبة للخصائص يتم ضبطها يدوياً من هذه النافذة بعد إدخال قيمة الخاصية أو إختيارها في الخانة المقابلة للأداة .. ويظهر تأثير تغيير قيمة الخاصية على الأداة غالباً في زمن التصميم وبشكل كلي في زمن التنفيذ .

= أما الأحداث فيتم إختيار الحدث بالنقر المزدوج عليه بحيث يظهر في نافذة الـCode على شكل دالة لها 2 parameters :

الأول sender من نوع الصنف object

الثاني e من نوع الصنف المشتق منه الحدث مثلاً EventArgs

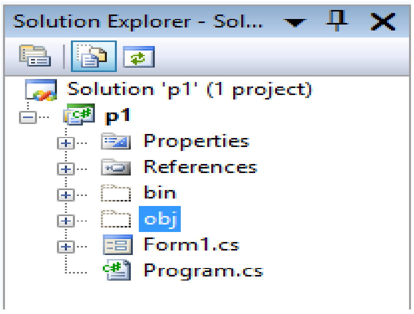
يتم كتابة الأكواد البرمجية داخل الحدث ويظهر تأثير الحدث على الأدوات بتنفيذ ما بداخله في زمن التنفيذ دائماً .

//الخصائص هي تأثيرات بصرية تحدث على الأداة بينما الأحداث هي أفعال المستخدم التي تتم على الأداة .

// عند التحديد على الخاصية أو الحدث في نافذة Properties فإنه تظهر عنها معلومات أسفل نافذة الخصائص .

// يمكن ضبط الخصائص للأداة برمجياً في نافذة الـCode كما يمكن إنشاء الحدث برمجياً على بنفس الشكل الذي يظهر به الحدث عند النقر عليه مع إضافة أكواد برمجية تقوم بتفعيل الحدث .

- نافذة Solution Explorer : وهي نافذة تضم الـProjects التي يتكون منها الـSolution والملفات التي يتكون منها كل Project ... وبالإمكان إضافة عناصر جديدة Items إلى الـProject وكذلك حذف عناصر من الـProject .. كما يمكن ضبط خصائص الـSolution&Project من هذه النافذة وعمليات أخرى مثل إعادة التسمية و Build و ...



// غالباً ما يكون إسم الـProject الأول بنفس إسم الـSolution مع إمكانية إعادة تسمية كليهما

// أي Project يتم إنشاؤه بشكل قياسي يتكون من عدة ملفات كالتالي :-

١- Properties : خصائص المشروع وتضم عدة ملفات كالتالي :

AssemblyInfo.cs : ملف مكتوب بلغة C# يتعامل مع لغة التجميع إسمبلي .

Resources.resx : ويضم ملف مكتوب بلغة C# عمصادر الملفات المستخدمة في الـProject بأنواعها المختلفة .

Settings.settings : ملف لضبط إعدادات الـProject .

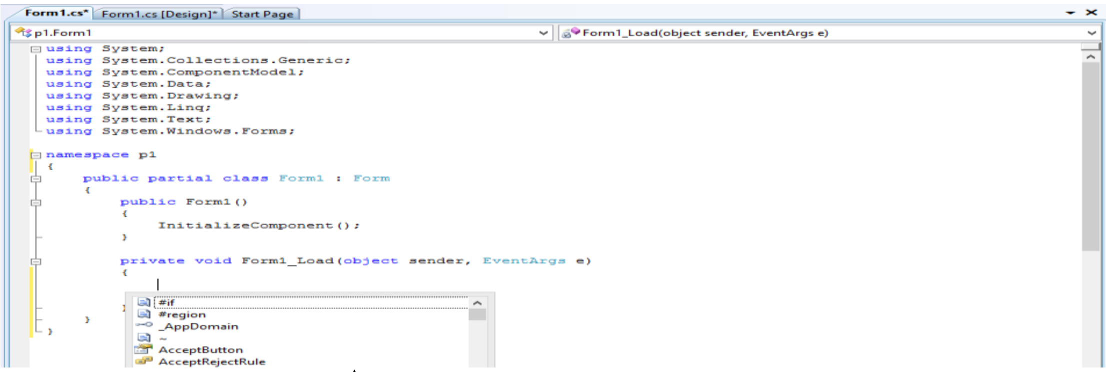
٢- References : ويضم جميع المكتبات والمراجع المستخدمة في الـProject .

٣- Form1.cs : ويضم عدة ملفات تخص التصميم المرئي للـForm التي يتكون منها الـProject ويمكن أن يحتوي الـProject أكثر من Form وهذه الملفات هي :

Form1.Designer.cs : ملف مكتوب بلغة C# عن أكواد التصميم المرئي وإضافة الأدوات بشكل يدوي وأكواد ضبط الخصائص بشكل يدوي …

Form1.resx : قالب مورد مبني XML .

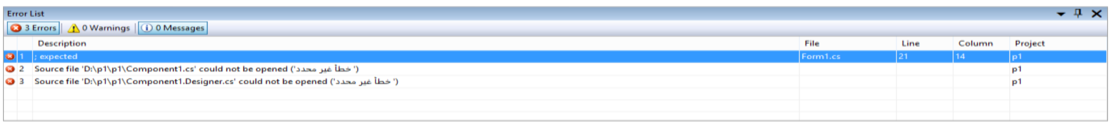
- Program.cs : ملف مكتوب بلغة C# يتم فيه تحديد الـForm التي يتم الإقلاع منها عند تنفيذ البرنامج ويعتبر الملف الرئيس للمشروع .
- نافذة الـCode: وهي شاشة المحرر التي يتم كتابة الأكواد البرمجية فيها بأي لغة من لغات Visual Studio ومنها لغة Visual C#.NET ويتميز هذا المحرر بخاصية الإكمال التلقائي للكود أي أنه بمجرد كتابة الحرف الأول من إسم الأداة أو الطريقة أو أي جملة فإنه تظهر قائمة تحتوي تلك الجملة وما شابهها..لاحظ الصورة التالية :(لإظهار قائمة الإكمال التلقائي إضغط Ctrl + Space)



// يمكن التنقل بين قائمة الإكمال التلقائي للكود بإستخدام مفاتيح الأسهم (↑) & (↓) وكذلك أيضاً بمجرد كتابة (.) خلف إسم الأداة مثلاً تظهر قائمة بخصائص وأحداث وطرق تلك الأداة .

// الكود الموجود في بداية شاشة الكود المسبوق بجملة يظهر بشكل تلقائي بمجرد إنشاء الـForm أما دوال الأحداث التي تظهر ويكتب الكود بداخلها تظهر بمجرد النقر على الحدث …

- نافذة Error List : وهي قائمة تضم الأخطاء البرمجية الموجودة في الأكواد البرمجية ووصف الخطأ ويظهر رقم السطر Line و رقم العمود Column الذي يوجد به الخطأ كما تتوفر إمكانية الذهاب إلى نافذة الـHelp حول حل ذلك الخطأ من مكتبة MSDN .. إنقر بالزر الأيمن على الخطأ تظهر قائمة إختر Show Error Help … لاحظ الصورة التالية :



- الإستثناءات Exceptions : الفرق بين Exception&Errorأنّ الـError هو خطأ برمجي بينما Exception خطأ منطقي مثل القسمة على الصفر أو عدد عناصر المصفوفة أكبر من حجمها

... إلخ .. في حال وجود Exception في البرنامج فإنه لن يتم إكمال تنفيذ البرنامج.. أما في حال وجود الـError فإنه يفشل عملية الـBuild مع إمكانية تنفيذ البرنامج ولكنه لا يعمل بشكل صحيح .

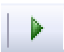
// يتم التغلب على مشكلة Exception من خلال عدة تراكيب أشهر أن نضع كود الإستثناء داخل جملة try .. catch كالتالي :

```
        try{
//Code
            }catch(Exception ex){
MessageBox.Show(ex.ToString());
                  }
```

- أزمنة (أطوار) البرنامج في لغة Visual C#.NET :

أولاً : مرحلة التصميم المرئي Design Time : ويتم فيه إنشاء الـForm وإضافة الأدوات يدوياً إليها ومن ثم ضبط خصائص تلك الأدوات يدوياً من نافذة الـProperties .

ثانياً : مرحلة البرمجة Programming Time : ويتم فيه كتابة الأكواد البرمجية بعد النقر على أحداث الأدوات داخل نافذة الـCode وبالإمكان معالجة الأحداث و ضبط خصائص الادوات برمجياً وإستخدام الطرق والأحداث للأداة وتعريف المتغيرات وإستخدامها وإنشاء طرق برمجيا وكتابة مختلف الأكواد البرمجية .

ثالثاً: مرحلة التنفيذ Run Time: يتم تنفيذ البرنامج بالنقر على الزر  أو الضغط على المفتاح F5وبالإمكان إدخال قيم تطبيق الأحداث وما بداخلها من أكواد ...

- مثال حول الأطوار التي يمر بها البرنامج في لغة VC#.NET :

- تطبيق ٢: صمم برنامج يقوم بإجراء العمليات الحسابية الأربع (+ ، - ، * ، /) بين عددين صحيحين؟

- قم بإضافة Project جديد واجعله بإسم "P2" ... تظهر الـForm1 تلقائياً

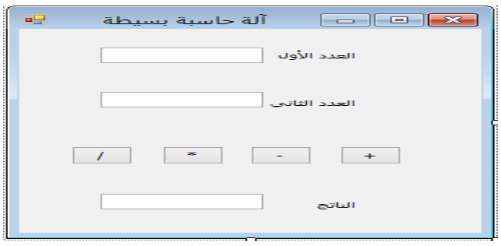
أولاً : نبدأ بمرحلة التصميم المرئي :

قم بإضافة الأدوات التالية إلى الـForm : ٣ من نوع TextBox– ٣ من نوع Label– ٤من نوع Button .. ثم قم بضبط الخصائص يدوياً كما في الجدول التالي :

م	الإسم البرمجي للأداة Name	الخاصية Text
١	Fom1	آلة حاسبة بسيطة
٢	label1	العدد الأول
٣	label2	العدد الثاني
٤	label3	الناتج
٥	button1	+

٦	button2	-
٧	button3	*
٨	Button4	/

قم بترتيب الأدوات على الـForm مثل ما هو موجود في الصورة التالية :-



ثانياً : نبدأ بمرحلة البرمجة : سنكتب الأكواد البرمجية داخل الحدث Click لأزرار الـButton حيث وهو الحدث الإفتراضي للأداة ..

- إنقر نقراً مزدوجاً على الزر + الذي أسمه البرمجي button1 يظهر حدث الـClick أكتب الكود:

```
privatevoid button1_Click(object sender, EventArgs e)
        {
textBox3.Text = (int.Parse(textBox1.Text) + int.Parse(textBox2.Text)).ToString();
        }
```

- عد إلى نافذة الـForm بالنقر بالزر الأيمن تظهر قائمة .. إنقر View Designer ..
- إنقر نقراً مزدوجاً على الزر - الذي أسمه البرمجي button2 يظهر حدث الـClick أكتب الكود:

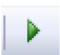
```
privatevoid button2_Click(object sender, EventArgs e)
        {
textBox3.Text = (int.Parse(textBox1.Text) - int.Parse(textBox2.Text)).ToString();
        }
```

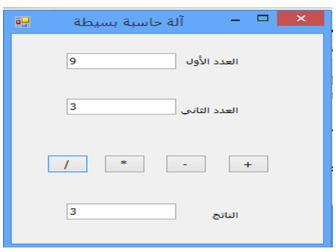
- إنقر نقراً مزدوجاً على الزر * الذي أسمه البرمجي button3 يظهر حدث الـClick أكتب الكود:

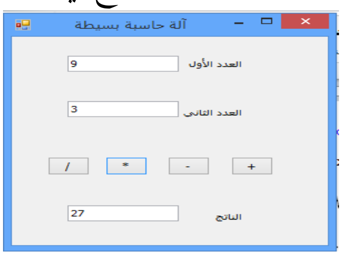
```
privatevoid button3_Click(object sender, EventArgs e)
        {
textBox3.Text = (int.Parse(textBox1.Text) * int.Parse(textBox2.Text)).ToString();
        }
```

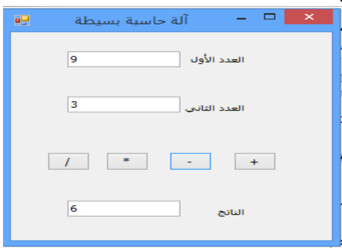
- إنقر نقراً مزدوجاً على الزر / الذي أسمه البرمجي button4 يظهر حدث الـClick أكتب الكود:

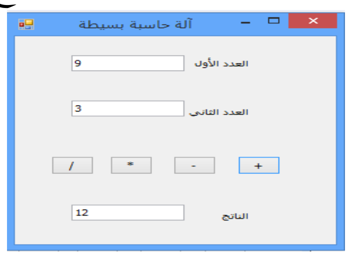
```
privatevoid button4_Click(object sender, EventArgs e)
        {
textBox3.Text = (int.Parse(textBox1.Text) / int.Parse(textBox2.Text)).ToString();
        }
```

ثالثاً : ننتقل الآن إلى زمن التنفيذ : إضغط الزر  تظهر النافذة .. قم بإدخال العدد الأول في مربع النص textBox1 مثلاً 9 ثم أدخل العدد الثاني في مربع النص textBox2 مثلاً 3 .. ثم إنقر على الزر + لاحظ الناتج .. إنقر الزر – .. إنقر الزر * .. أنقر الزر / .. لاحظ الناتج في كل مرة









- قم بالخروج من زمن التنفيذ بالنقر على الزر  .. قم بحفظ البرنامج بالنقر على الزر  .

الفصل الرابع

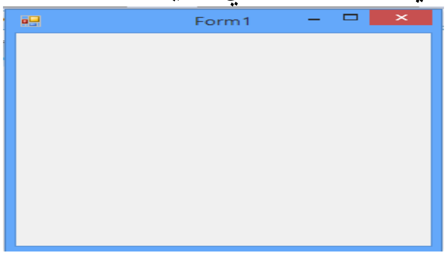
التعامل مع النماذج في لغة VC#.NET

Managing Forms In Visual C#.NET

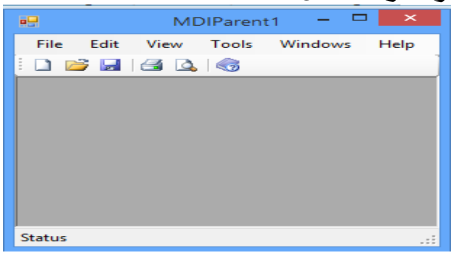
- النموذج الـForm : تسمى النافذة في زمن التصميم Form أما في زمن التنفيذ تسمى Window ، لكن الاصطلاح الشائع للنافذة في كلا الزمنين حالياً تسمى Form .

● أنواع الـForm : توجد ٣ أشكال من الـForm في لغة Visual C#.NET :

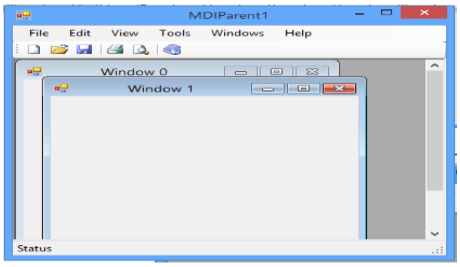
١- Standard Form النموذج القياسي : وهي الشكل الإفتراضي الذي يظهر عند إنشاء مشروع جديد وهو الأكثر إستخداماً



٢- MDI Form النماذج متعددة الوثائق : وتظهر بالشكل :



٣- MDI child Form النماذج الأبناء تظهر غالباً داخل الـMDI Parent بالشكل :



● أجزاء ومكونات الـForm :

١- Title Bar شريط عنوان الـForm : يقع في أعلى الـFormويتكون من التالي :

- Control Box صندوق التحكم ويتكون من ٣ أزرار :

close button زر الإغلاق – maximizing buttزر التكبير–

minimizing button زر التصغير .

- عنوان الـForm : هو الإسم الظاهري للـForm ويقع وسط شريط العنوان Form1 .

- أيقونة الـForm : هي صورة موجودة يسار شريط العنوان عند تصغير الـForm فإنها تظهر على شريط المهام .

٢- Client Area منطقة الزبون : تقع وسط الـForm وهي المنطقة التي يتم إضافة الأدوات Controls عليها ويأخذ المبرمج حريته بالعمل عليها في زمن التصميم المرئي ...

٣- الـForm Border حدود النموذج : يتمثل في الإطار الذي يحيط بالـForm ...

- خصائص الـForm : حدد على الـForm لكي تمتلك التركيز .. إنتقل إلى نافذة الخصائص لضبط خصائص الـForm يدوياً وكذلك يمكن ضبط الخصائص برمجياً .. سنتناول أهم الخصائص كالتالي :

- Name : الإسم البرمجي للنموذج والاسم الإفتراضي ....Form1,Form2 ويستخدم هذا الإسم عند كتابة الأكواد البرمجية ويمكن تعديل الـName بحيث يدل على الوظيفة التي تؤديها الـForm ويفضل أن يسبق بـfrm مثلاً frmfrist ... وتستقبل وتعيد قيمة من نوع String ولضبط الـName برمجيا:: `this.Name = "frmfrist";`
- Text : عنوان الـForm وهو الإسم الذي يظهر على شريط العنوان وتستقبل وتعيد قيمة من نوع String ولضبط الـText برمجياً : `this.Text = " Visual C#.NET ";`
- Icon : أيقونة الـFormيمكن تغيير صورة أيقونة الـForm من خلال هذه الخاصية بشرط أن تكون الصورة من نوع (بإمتداد) ico. إنقر على الزر 

تظهر نافذة حدد ملف الصورة من المكان الموجود فيه على جهاز الكمبيوتر ثم إنقر الزر Open .. لاحظ تغيير أيقونة الـForm .. ولتغيير أيقونة الـForm برمجياً : ضع ملف الصورة في القرص Dبإسم a1.ico إكتب الكود كالتالي :

```
this.Icon = Icon.ExtractAssociatedIcon("d:\\a1.ico");
```

وفي حال نسخ ملف الصورة a1.ico في مجلد الـDebug داخل مجلد البرنامج يكتب الكود :

```
this.Icon = Icon.ExtractAssociatedIcon("a1.ico");
```

- ShowIcon : عندما تكون القيمة True تظهر أيقونة الـForm وعندما تكون القيمة False تختفي أيقونة الـForm .. تستقبل وتعيد قيمة من نوع Boolean .. لإخفاء أيقونة الـForm برمجياً : `this.ShowIcon = false;`

- ShowInTaskbar: للتحكم في ظهور أيقونة الـForm على شريط المهام ، عندما تكون True تظهر وعندما تكون False تختفي عند تصغير الـForm .. تستقبل وتعيد قيمة من نوع Boolean .. لإخفاء أيقونة الـForm من شريط المهام برمجياً :

```
this.ShowInTaskbar = false;
```

- ControlBox : للتحكم في ظهور صندوق التحكم على شريط العنوان للـForm ، عندما تكون True يظهر وعندما تكون False يختفي من شريط العنوان .. تستقبل وتعيد قيمة من نوع Boolean .. لإخفاء صندوق التحكم من شريط عنوان الـForm برمجياً :

```
this.ControlBox = false;
```

- MaximizeBox : للتحكم في ظهور زر التكبير في صندوق التحكم ، في حال True يظهر أما في حال False يختفي .. تستقبل وتعيد قيمة من نوع Boolean .. لإخفاء زر التكبير برمجياً :

```
this.MaximizeBox = false;
```

- MinimizeBox : للتحكم في ظهور زر التصغير في صندوق التحكم ، في حال True يظهر أما في حال False يختفي .. تستقبل وتعيد قيمة من نوع Boolean .. لإخفاء زر التصغير برمجياً :

```
this.MinimizeBox = false;
```

- RightToLeft : تستقبل وتعيد قيمة من نوعRightToLeft وتحمل ٣قيم كالتالي :

No : محاذاة عنوان الـForm من اليسار إلى اليمين .

Yes : محاذاة عنوان الـForm من اليمين إلى اليسار .

Inherit : يستخدم في حالة الوراثة .

- RightToLeftLayout : تستقبل وتعيد قيمة من نوع Boolean ، في حال True يتم محاذاة صندوق التحكم من اليمين إلى اليسار وفي حال False تبقى كما هي بدون محاذاة ..

للحصول على الواجهة العربية بحيث تكون الأيقونة في اليمين وصندوق التحكم في اليسار قم بضبط الخاصية RightToLeft بالقيمة Yes والخاصية RightToLeftLayout بالقيمة True ويمكن الحصول على الواجهة العربية برمجياً :

```
this.RightToLeft = RightToLeft.Yes;
this.RightToLeftLayout = true;
```

- FormBorderStyle : نمط حدود الـForm .. تستقبل وتعيد قيمة من نوع FormBorderStyle وتحمل ٧قيم كالتالي :

None : بلا حدود .

FixedSingle : حدود ثابتة تتكون من خط واحد .

Fixed3D : حدود ثابتة ثلاثية الأبعاد .

FixedDialog : حدود سميكة وثابتة من نمط حدود مربعات الحوار .

Sizable : حدود تقبل أن يتغير حجمها .

FixedToolWindow : حدود نافذة أدوات ثابتة لا تقبل تغيير حجمها .

SizableToolWindow : حدود نافذة أدوات تقبل بأن يتغير حجمها .

لجعل حدود الـForm ثابتة ثلاثية الأبعاد برمجياً :

```
this.FormBorderStyle = FormBorderStyle.Fixed3D;
```

- BackColor : لون خلفية الـForm .. تستقبل وتعيد قيمة من نوع Color .. توجد لدينا ٣ مجموعات ألوان في Visual Studio وهي :

System : ألوان النظام (يقصد به نظام التشغيل Windows تقريباً) .

Web : ألوان مواقع الويب .

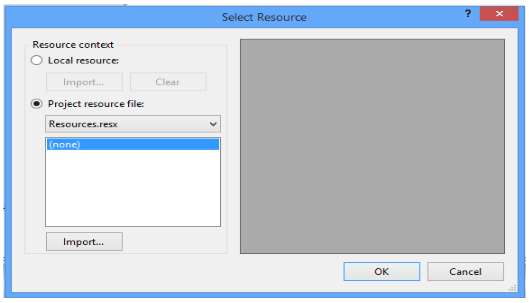
Custom : ألوان مخصصة .

اللون الإفتراضي للـForm هو Control ثم بتغيير لون الخلفية إلى الأحمر .. لضبط لون خلفية الـForm إلى اللون الأحمر برمجياً :

```
this.BackColor = Color.Red;
```

- BackgroundImage : تعيين صورة خلفية للـForm 

إنقر على الزر المقابل للخاصية .. تظهر نافذة Select Resource إنقر على الزر Import .. حدد ملف الصورة ثم إنقر Open .. لاحظ إدراج الصورة للنافذة .. إنقر Ok .



نوع (إمتداد) الصورة التي تضاف كخلفية للـForm :

JPEG,PNG,GIF, ...

تستقبل وتعيد هذه الخاصية قيمة من نوع Image

لتغيير صورة خلفية الـForm برمجياً :

ضع ملف الصورة في القرص D بإسم a2.jpg .. إكتب الكود التالي :

```
this.BackgroundImage = Image.FromFile("d:\\a2.jpg");
```

وإذا تم وضع الصورة في مجلد Debug داخل مجلد البرنامج يكتب الكود كالتالي :

```
this.BackgroundImage = Image.FromFile("a2.jpg");
```

- BackdroundImageLayout : نمط تموضع صورة الخلفية على الـForm .. تستقبل وتعيد قيمة من نوع ImageLayout وتحتمل ٥ قيم كالتالي :

None : تظهر الصورة بحجمها الأصلي إبتداءً من أعلى يسار الـForm .

Tile : تظهر الصورة بحجمها الأصلي بأكثر من نسخة على الـForm .

Center : تظهر الصورة بحجمها الأصلي وسط الـForm .

Stretch : يكون حجم الصورة بنفس حجم الـForm .

Zoom : تكبير حجم الصورة أكبر ما يمكن تموضع وسط الـForm .

لجعل صورة خلفية الـForm بنفس حجم الـForm برمجياً :

```
this.BackgroundImageLayout = ImageLayout.Stretch;
```

- Size : حجم الـForm ويتكون من خاصيتين هما Width العرض ، Heightالإرتفاع وكلا الخاصيتين تستقبل وتعيد قيمة من نوع int .. لضبط حجم الـForm برمجياً :

```
this.Size = newSize(500, 500);
```

- Location : موقع الـForm على الشاشة وتتكون من خاصيتين هما X محور السينات (الأعمدة) ، Y محور الصادات (الصفوف) وكلا الخاصيتين تستقبل وتعيد قيمة من نوع int .. لضبط موقع الـForm على الشاشة برمجياً :

```
Location = newPoint(200, 200);
```

- StartPosition : الموقع الإبتدائي الذي يتم إقلاع الـForm منه بالنسبة للشاشة عند تنفيذ البرنامج .. يستقبل ويعيد قيمة من نوع FormStartPosition ويحتمل ٥ قيم كالتالي :

Manual : تحدد الخاصيتان Size ، Location موقع الـForm الإبتدائي .

CenterScreen : يظهر الـForm في وسط الشاشة وتحدد أبعاده الخاصية Size .

WindowsDefaultLocation : يعرض الـForm في الموقع الإفتراضي المحدد من قبل النظام Windows وتؤخذ الأبعاد المحددة في الخاصية Sizeله .

WindowsDefaultBounds : يعرض النموذج في الموقع الإفتراضي المحدد من قبل النظام Windows وتؤخذ الأبعاد المحددة في النظام Windows بشكل إفتراضي .

CentetParent : يظهر الـForm في وسط حدود الـForm الأب الذي قام بإستدعائه .

لجعل الـForm يقلع عند التنفيذ وسط الشاشة برمجياً :

```
this.StartPosition = FormStartPosition.CenterScreen;
```

- WindowState : حجم الـForm على الشاشة بعد تنفيذ البرنامج .. تستقبل وتعيد قيمة من نوع FormWindowsState وتحتمل ٣ قيم كالتالي :

Normal : تعرض الـForm إلى الحالة الطبيعية .

Minimized : تعرض الـForm مصغرة على شكل أيقونة .

Maximized : تعرض الـForm مكبرة إلى أقصى حد .

لجعل الـForm أكبر ما يمكن بعد التنفيذ برمجياً :

```
this.WindowState = FormWindowState.Maximized;
```

- Locked : تقوم بإقفال الـForm بحيث لا يمكن إعادة تحجيمها في زمن التصميم بواسطة المؤشر على حواف الـForm .. تستقبل وتعيد قيمة من نوع Boolean .. عندما تكون القيمة True إقفال وعندما تكون False تحرير .. تضبط هذه الخاصية يدوياً فقط .

- توجد عدة خصائص أخرى للـForm سيتم شرحها ضمن تطبيقات برمجية لاحقاً .

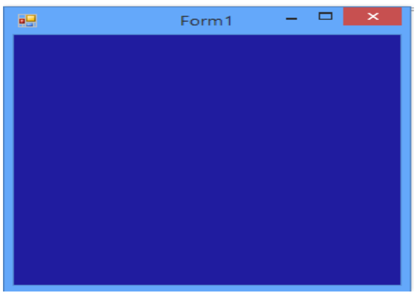
- أهم أحداث الـForm :
  - Load : يحدث عند إقلاع الـForm أو عندما تعرض الـForm لأول مرة .
  - Activated : يحدث عند تنشيط (تفعيل) الـForm بواسطة Code أو من قبل المستخدم .
  - Click : يحدث عند النقر على سطح الـForm .
  - Closed : يحدث عندما يكون الـForm قد أغلق .
  - Closing : يحدث عند إغلاق الـForm .
  - Paint : يحدث عند إعادة رسم الـForm .
  - Resize : يحدث عند إعادة تحجيم (تغيير أبعاد) الـForm .
  - توجد أحداث أخرى للـForm مثل أحداث الفأرة وأحداث لوحة المفاتيح ... سيتم تناولها بالتفصيل لاحقاً .

- أهم طرق الـForm :
  - Close : تقوم بإغلاق الـForm .
  - Focus : إعطاء الـForm التركيز .
  - Hide : تقوم بإخفاء الـForm .
  - SetBounds : تضبط أبعاد وموقع الـForm (تقوم بعمل خاصيتي Size ، Location معاً) .
  - Show : تقوم بإظهار الـForm .

  // توجد طرق أخرى للـForm سيتم تناولها ضمن التطبيقات البرمجية ...

- تطبيق٣ : صمم برنامجاً يقوم بتغيير لون خلفية الـForm كل ثانية بشكل عشوائي ؟
  - قم بإنشاء Project جديد بإسم P3 .. تظهر نافذة الـForm قم بإضافة أداة Timer ثم إضبط خصائص الـTimer يدوياً كالتالي: الخاصية Enabled بالقيمة True& الخاصية Inerval بالقيمة 1000 ملي ثانية .. في حدث الـTickللـtimer1 إكتب الكود بالشكل التالي :

```
privatevoid timer1_Tick(object sender, EventArgs e)
        {
Random r = newRandom();
this.BackColor = Color.FromArgb(r.Next(0, 255), r.Next(0, 255), r.Next(0, 255));
        }
```



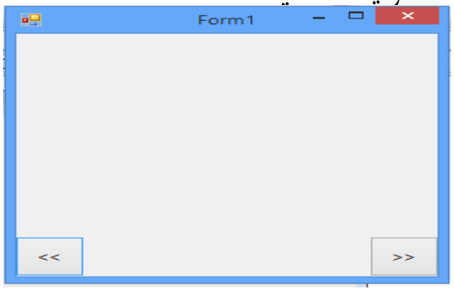
قم بتنفيذ البرنامج .. لاحظ تغيير لون الـForm كل ثانية من الزمن ..

// الطريقة Next من الصنف Random تولد أرقام عشوائية صحيحة .. والطريقة FromArgb من الصنف Color تخلط الألوان

Blue&Green&Red وفق القيم المدخلة لتولد لون جديد .

- تطبيق ٤ : صمم برنامجاً يقوم بعرض ٥ صور مثلاً كخلفية للForm .. يتم التنقل بين الصور بالنقر على زر التالي يتوقف عند آخر صورة وزر السابق يتوقف عند أول صورة ؟

- قم بإنشاء Project جديد بإسم P4 .. تظهر الـForm1 قم بإضافة زرين Button .. إجعل قيمة الخاصية Textللbutton1 "<<" وقيمة الخاصية Textللbutton2 ">>" .. لاحظ الشكل :



- إذهب إلى مجلد الصور ثم قم بنسخ ٥ صور في القرص D .. قم بإعادة تسمية الصور بالأرقام كالتالي : الصورة الأولى "1" ، الصورة الثانية "2"، ...وهكذا حتى الصورة الخامسة "5"

- قم بتعريف متغير i وأعطه قيمة إبتدائية 1 كما يلي :

```
publicpartialclassForm1 : Form
    {
int i = 1;
```

- في الحدث Loadللform1 إكتب الكود بالشكل :

```
privatevoid Form1_Load(object sender, EventArgs e)
        {
this.BackgroundImage = Image.FromFile("d:\\" + i.ToString() + ".jpg");
this.BackgroundImageLayout = ImageLayout.Stretch;
        }
```

- في حدث Clickللbutton1 إكتب الكود بالشكل :

```
privatevoid button1_Click(object sender, EventArgs e)
        {
            i++;
if (i >= 5)
                i = 5;
this.BackgroundImage = Image.FromFile("d:\\" + i.ToString() + ".jpg");
this.BackgroundImageLayout = ImageLayout.Stretch;

        }
```

- في حدث Clickللbutton2 إكتب الكود بالشكل :

```
privatevoid button2_Click(object sender, EventArgs e)
        {
            i--;
if (i <= 1)
                i = 1;
this.BackgroundImage = Image.FromFile("d:\\" + i.ToString() + ".jpg");
ackgroundImageLayout = ImageLayout.Stretch;

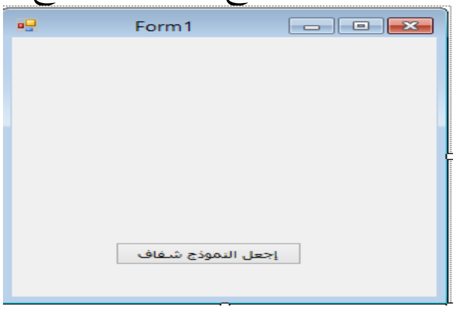
        }
```



- قم بتنفيذ البرنامج .. ثم تنقل بين الصور بالنقر على الأزرار .. إستمتع

تطبيق ٥ : صمم برنامجاً يجعل الـForm شفاف بحيث يمكننا رؤية النوافذ والمجلدات والعناصر الموجودة خلفه سواءً في سطح المكتب أو أي نافذة أخرى ؟

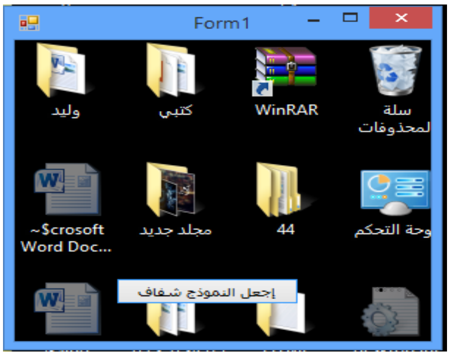
- قم بإنشاء Project جديد بإسم P5 .. تظهر نافذة الـForm .. أضف أداة زر الـButton إليها واجعل خاصية الـText له بالقيمة "إجعل النموذج شفاف " .. يصبح الـForm بالشكل :



- في حدث الـClickللـbuton1 إكتب الكود بالشكل :-

```
privatevoid button1_Click(object sender, EventArgs e)
        {
this.BackColor = Color.Wheat;
this.TransparencyKey = Color.Wheat;
    }
```

- قم بتنفيذ البرنامج .. إضغط على الزر (إجعل النموذج شفاف) .. لاحظ رؤية الأشياء الموجودة خلف النموذج ..



الفصل الخامس

التعامل مع النماذج المتعددة

Working WithMultiple Forms

لقد نظمت مشاريع Windows في VC# بتوزيعها إلى نماذج وأصناف ووحدات برمجية فكيف يمكن أن تصل الشيفرة في أحد الأجزاء إلى الجزء الآخر .. في هذا الفصل سيتم تطبيق برمجيات لمشاريع تحتوي على أكثر من Form في وقت واحد وكيف يتم التنقل فيما بينها والتعامل مع أشكال متنوعة من النماذج المتعددة …

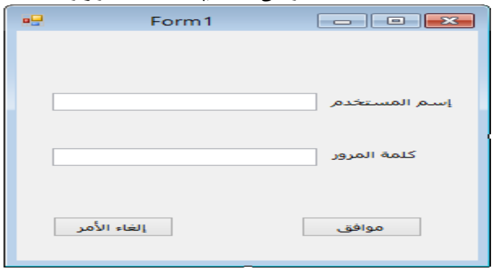
- أولاً : مشاريع تحتوي على أكثر من Form من النوع القياسي :

- تطبيق ٦ : صمم برنامج إسم المستخدم وكلمة المرور بحيث يكون الـForm1 يقوم بفحص إسم المستخدم وكلمة المرور في حال كانت صحيحة يتم الإنتقال إلى الـForm2 اما في حالة كانت خاطئة فإنه تظهر رسالة تخبر المستخدم بأن إسم المستخدم أو كلمة المرور خاطئة … يتيح البرنامج ٣ فرص محاولات في حال إنتهاء عدد المحاولات يتم الخروج من البرنامج بشكل نهائي تلقائياً ؟

- قم بإنشاء Project جديد بإسم P6 .. تظهر الـForm1 قم بإضافة الأدوات التالية إليها : ٢ أداة Label ، ٢ أداة TextBox ، ٢ أداة Button وأضبط الخصائص كما في الجدول :

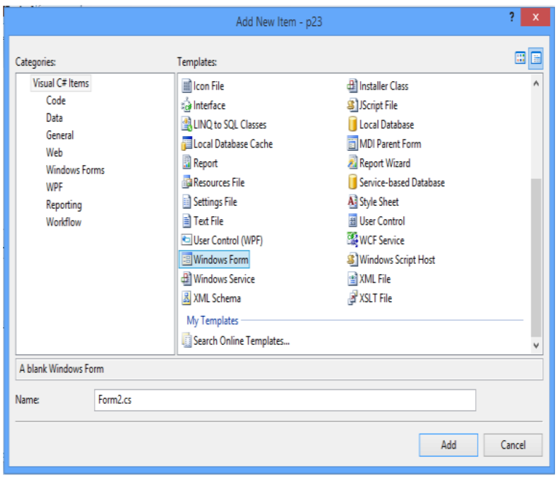
الأداة	الخاصية Text
label1	إسم المستخدم
label2	كلمة المرور
button1	موافق
button2	إلغاء الأمر

إضبط خاصية PasswordChar للـtextBox2 بالقيمة ' * ' لغرض تشفير كلمة المرور .. يصبح شكل الـForm1 كما في الصورة :



- قم بإضافة Form آخر إلى الـProject .. من القائمة Project ـــ>Add WindowsForm..

تظهر نافذة Add New Item .. من المجموعة Templates إختر Windows Form ويظهر الإسم الإفتراضي Form2.cs إنقر على الزر Add .

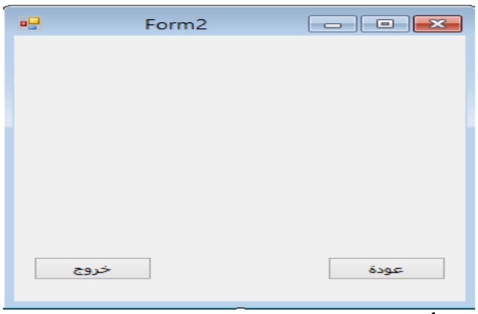


لاحظ إضافة نموذج جديد بإسم Form2

- قم بإضافة ٢ أداة Button1 وأضبط الخاصية Text كالتالي :

button1 ـ> "عودة"&button2 ـ> "خروج"

تظهر الـForm2 كما في الصورة التالية :



- عُد إلى Form1 لكتابة الأكواد البرمجية ... إنقر بالزر الأيمن على سطح الـForm1 ثم إختر View Code ... في قسم التصريحات إكتب الكود بالشكل التالي :-

```
publicpartialclassForm1 : Form
    {
int i = 1;
Form2 f2 = newForm2();
```

قمنا بتعريف i الذي يمثل عدد المحاولات والـ f2 كائن (نسخة) من الـForm2 ..

- في حدث الـClickللbutton1 (موافق) إكتب الكود بالشكل التالي :-

```
privatevoid button1_Click(object sender, EventArgs e)
        {
if (i <= 3)
if (textBox1.Text == "ali"&& textBox2.Text == "1234")
                {
this.Hide();
                    f2.Show();
                }
else
                {
   i = i + 1;
MessageBox.Show("إسمالمستخدمأوكلمة المرورغيرصحيح ");
   textBox1.Text = "";
   textBox2.Text = "";
   textBox1.Focus();
                }
else
            {
MessageBox.Show("لقد أستنفذتعددالمحاولات .. لذا سيتم الخروج من البرنامج ");
this.Close();
            }

        }
```

- في حدث الـClickللbutton2(إلغاء الأمر) نكتب الكود بالشكل التالي :-

```
privatevoid button2_Click(object sender, EventArgs e)
        {
this.Close();
        }
```

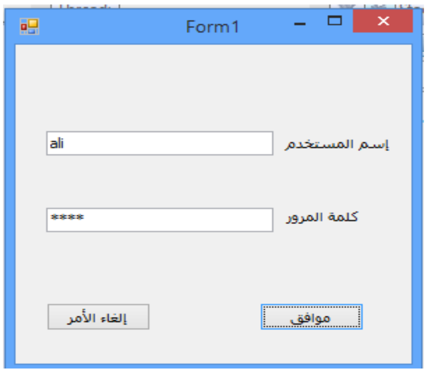
- عُد إلى الـForm2 لكتابة الأكواد البرمجية .. في حدث الـClickللbutton1 (عودة) نكتب الكود بالشكل :-

```
privatevoid button1_Click(object sender, EventArgs e)
        {
Form1 f1 = newForm1();
            f1.Show();
this.Hide();
        }
```

- في حدث الـClickللbutton2 نكتب الكود بالشكل التالي :-

```
privatevoid button2_Click(object sender, EventArgs e)
        {
            System.Environment.Exit(0);
        }
```

- قم بتنفيذ البرنامج .. أدخل إسم المستخدم وكلمة المرور بشكل صحيح ثم إنقر موافق .. يتم الدخول للـForm2 .. نفذ البرنامج مرة أخرى وأدخل إسم المستخدم أو كلمة المرور بشكل خاطئ بأكثر من ٣ مرات ولاحظ ماذا يحدث ...



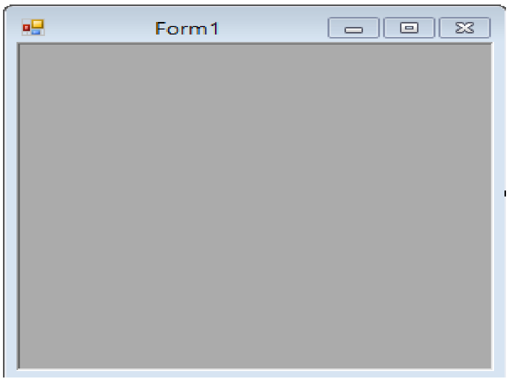
- ثانياً : إنشاء تطبيقات الواجهات متعددة الوثائق MDI :

يمكن لنوافذ إطار MDI أن تعرض عدة نوافذ أبناء داخلها وبالإمكان ترتيب النوافذ الأبناء داخل الـForm الأب بأشكال مختلفة ...

- تطبيق ٧ : قم بإنشاء نافذة متعددة الوثائق MDI وقم بترتيبها بكل أنواع التراتيب المتاحة على تكون الأوامر ضمن قوائم Menu ؟

- قم بإنشاء Project جديد بإسم P7 .. تظهر الـForm1 ..

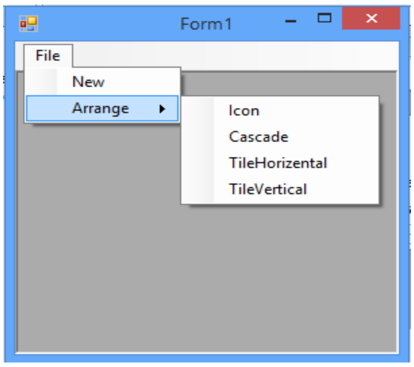
إضبط الخاصية IsMdiContainer بالقيمة True لجعل الـForm1 من النوع الحاوي (الأب) لاحظ تغير لون الـForm1 بالشكل :



- قم بإضافة أداة MenuStrip إلى الـForm1 من صندوق الأدوات يظهر شريط قوائم أعلى الـForm1 .. لإضافة بنود رئيسية (قوائم) في مربع النص " Type Here " إكتب File لاحظ ظهور مربع" Type Here " أسفل الـFile وبجانبه أيضاً من أجل القائمة التالية ...

- أضف بند جديد في القائمة File بإسم New في مربع النص Type Here أسفل الـFile ...
- ضمن القائمة File قم بإنشاء بند جديد بإسم Arrange وأضف إليها بنود فرعية أربعة كالتالي :

TileVertical - TileHorizental–Cascade - Icon



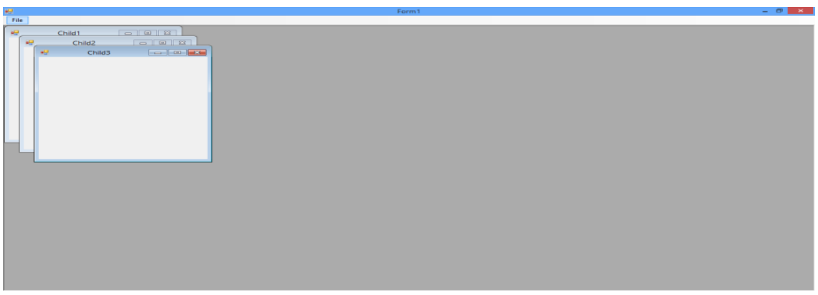
- أضف Form جديد إلى المشروع بإسم Form2 من Project<ـــ Add WindowsForm
- عُد إلى الـForm1 لكتابة الأكواد البرمجية ... في حدث الـLoadللـForm1 إكتب الكود بالشكل :

```
privatevoid Form1_Load(object sender, EventArgs e)
        {
this.WindowState = FormWindowState.Maximized;
        }
```

- إنقر نقراً مزدوجاً على البند New واكتب الكود بالشكل :

```
privatevoid newToolStripMenuItem_Click(object sender, EventArgs e)
        {
Form2 f = newForm2();
            f.MdiParent = this;
            f.Text = "Child" + MdiChildren.Length.ToString();
            f.Show();
        }
```

// الخاصية MdiParent تحدد من هو الـForm الحاوي (الأب) .. أما الخاصية MdiChildrenللـLength تعيد عدد الـForm الأبناء فتظهر تسمية النماذج الأبناء Child1،Child2 وهكذا .. قم بتنفيذ البرنامج وأنقر أكثر من مرة على البند New مع كل نقرة ينشأ إبنForm ..



- عُد إلى زمن التصميم والبرمجة .. إنقر على البند الفرعي Icon المتفرع من البند Arrange وأكتب الكود بالشكل :

```
privatevoid iconToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.ArrangeIcons);        }
```

وتعني القيمة ArrangeIcons : ترتيب كل أيقونات الـMDIالأبناء بحيث تعرض هذه الأيقونات عند تصغير نافذة الـMDI .

- إنقر نقراً مزدوجاً على البند الفرعي Cascade المتفرع من البند Arrange واكتب الكود بالشكل :

```
privatevoid cascadeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.Cascade);
        }
```

وتعني القيمة Cascade : ترتيب كل نوافذ الـMDIالأبناء بشكل متتالي إعتباراً من الزاوية اليسارية العليا لمنطقة الزبون .

- إنقر نقراً مزدوجاً على البند الفرعي TileHorizental المتفرع من البند Arrange واكتب الكود بالشكل :

```
privatevoid tileHorizentalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.TileHorizontal);
       }
```

وتعني القيمة TileHorizental : ترتيب كل نوافذ الـMDI الأبناء بحيث تتجاور أفقياً وتغطي كل منطقة الزبون ولا تتداخل فيما بينها .

- إنقر نقراً مزدوجاً على البند الفرعي TileVertical المتفرع من البند Arrange واكتب الكود بالشكل :

```
privatevoid tileVerticalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            LayoutMdi(MdiLayout.TileVertical);
        }
```

وتعني القيمة TileVertical : ترتيب كل نوافذ الـMDI الأبناء بحيث تتجاور عمودياً وتغطي كل منطقة الزبون ولا تتداخل فيما بينها .

- قم بتنفيذ البرنامج .. أنشأ ٤ نوافذ أبناء بالنقر على البند New .. قم بترتيبها من البند Arrange وجرب مختلف أشكال التراتيب المتفرعة منه ... إستمتع .
- ثالثاً : إنشاء الـForm على شكل مربعات حوار مع تعدد الـForm :

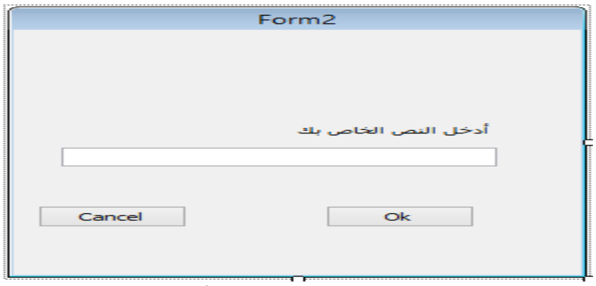
- تطبيق ٨ : صمم برنامج يجعل الـForm على شكل مربع حوار بحيث يتم إرسال النص من مربع نص في الـForm2 إلى مربع نص في الـForm1 بمجرد النقر على الزر Ok ؟
  - قم بإنشاء Project جديد بإسم P8 .. تظهر نافذة الـForm1 ...
  - قم بإضافة Form آخر بإسم Form2 من القائمة Project ــ Add WindowsForm> ..
  - قم بإضافة الأدوات التالية إلى Form2 :

أداة Label وأجعل خاصية الـText بالقيمة " أدخل النص الخاص بك" ... ثم أضف أسفل من الـlabel1 أداة textBox .. ثم أضف ٢ أزرار Button واضبط الخاصية Textللbutton1 بالقيمة " Ok " وأضبط الخاصية Textللbutton2 بالقيمة "Cancel " ...

- لكي يظهر الـForm2 على شكل مربعات الحوار إضبط الخصائص التالية كما في الجدول :

الخاصية	القيمة
FormBorderStyle	FixedDialog
ControlBox	False
ShowInTaskbar	False

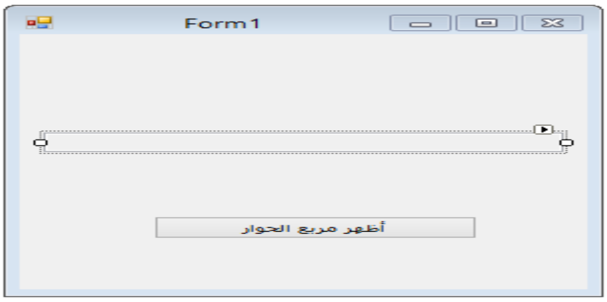
بعد ضبط الخصائص السابقة للـForm2 يظهر بالشكل :



- إضبط الخاصية DialogResult للزرين Cancel&Ok كما في الجدول :-

الأداة	قيمة الخاصية DialogResult
button1	OK
button1	Cancel

- لكي نضمن نقل المعلومات من بين النماذج Form2 ــForm1> نجعل الخاصية Modifiers لمربع النص textBox1 الموجود في Form2 بالقيمة Public .
- لعرض وقراءة المعطيات من مربع الحوار نعود إلى الـForm1 ونقوم بالتالي :-
قم بإضافة أداة زر Button بإسم button1 وأجعل الخاصية Text بالقيمة "أظهر مربع الحوار" أضف أداة مربع النص TextBox بإسم textBox1 واجعل الخاصية ReadOnly بالقيمة True .. يظهر بعدها الـForm1 بالشكل :



- سنقوم الآن بالبرمجة في الـForm1 إنقر بالزر الأيمن واختار View Code ... في قسم التصريحات ننشئ Object من الـForm2 يظهر الكود بالشكل :

```
publicpartialclassForm1 : Form
    {
Form2 f = newForm2();
```

- في الحدث Click للـbutton1 نكتب الكود بالشكل :

```
privatevoid button1_Click(object sender, EventArgs e)
        {
```

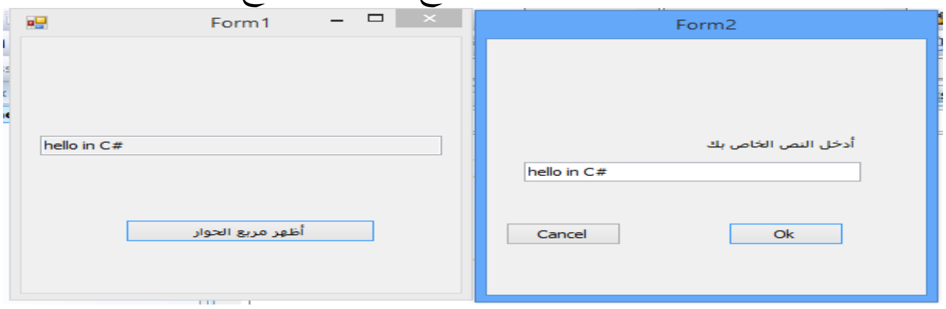
```
if (f.ShowDialog() == DialogResult.OK)
                textBox1.Text = f.textBox1.Text;
        }
```

// الطريقة ShowDialog تتعامل مع الـForm من نوع مربعات الحوار وتعيد قيمة من نوع DialogResult تدل على الزر الذي تم النقر عليه Ok أو Cancel .

- ننتقل إلى الـForm2 .. للتحكم بالزرين Cancel&Ok من لوحة المفاتيح بحيث عند الضغط على المفتاح Enter ينفذ الزر Ok وعند الضغط على المفتاح ESC ينفذ الزر Cancel وذلك عندما يمتلك الـForm2 التركيز .. في الحدث Form2_Load نكتب الكود بالشكل :

```
privatevoid Form2_Load(object sender, EventArgs e)
        {
this.AcceptButton = button1;
this.CancelButton = button2;
        }
```

- قم بتنفيذ البرنامج .. إنقر على الزر "أظهر مربع الحوار" تظهر Form2 .. إكتب نصاً داخل مربع النص مثلاً " Hello in C# "ثم إضغط الزر Ok أو إضغط المفتاح Enter ..
لاحظ إرسال النص من الـForm2إلى مربع النص الموجود في الـForm1 ...
كرر العملية أكثر من مرة ... جرب الزر Cancel أو المفتاح ESC ... استمتع



- رابعاً : إنشاء النموذج المملوك Owned Form
النموذج المملوك عبارة عن Form يتم التحكم بها من Form أخرى بحيث عند تصغير الـForm المالك على شريط المهام فإن الـForm المملوك يتم تصغيره تلقائياً معه وعند إعادته فإنه يعود معه

- تطبيق ٩ : صمم برنامج لـForm مملوك لـForm آخر ... ؟
- قم بإنشاء Project جديد بإسم P9 .. تظهر نافذة الـForm1
- قم بإضافة Form آخر بإسم Form2 من القائمة Project ــ Add Windows Form...>
- في الـForm1 أضف أداة زر Button بإسم button1 وأجعل الخاصية Text له بالقيمة : "أظهر النموذج المملوك" .
- في الـForm2 أضف أداة Label بإسم label1 وأجعل الخاصية Text بالقيمة : "النموذج المملوك" .
- سننتقل الآن إلى نافذة الـCode في الـForm1 .. في قسم التصريحات قم بكتابة الكود بالشكل :

```
publicpartialclassForm1 : Form
    {
Form2 f = newForm2();
```

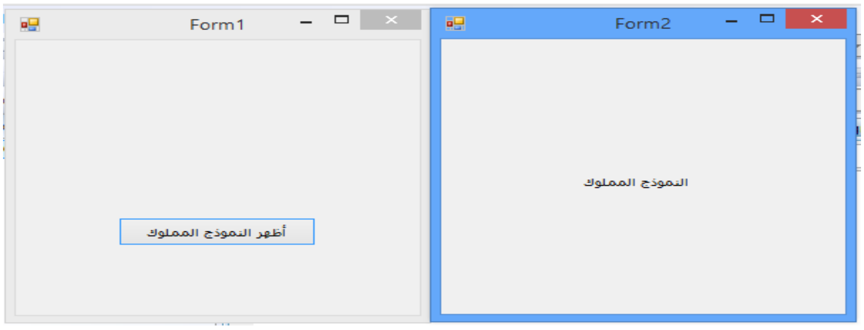
- في الحدث button1_Click نكتب الكود بالشكل :

```
privatevoid button1_Click(object sender, EventArgs e)
```

```
            {
                f.Show();
    this.AddOwnedForm(f);
            }
```

// الطريقة AddOwnedForm تجعل الـForm الموجود في البارامتر للطريقة مملوك للـForm الحالي .

- قم بتنفيذ البرنامج .. إنقر على الزر "أظهر النموذج المملوك" تظهر الـForm2 .. قم بتصغير الـForm1 على شريط المهام لاحظ تصغير الـForm2 معه .. قم بإعادة الـForm1 إلى وضعه السابق على الشاشة لاحظ عودة الـForm2 معه ...



- رابعاً : إنشاء نماذج تبقى دائماً في المقدمة :
يمكنك أن تجعل الـForm في زمن التنفيذ فوق باقي النماذج بضبط الخاصية TopMost له بالقيمة True .. كما نستطيع أن نغير ترتيب تتالي نماذجك (الـForms) في زمن التنفيذ بإستخدام الطريقتين BringToFront (إحضار إلى الأمام) ، SentToBack (إرسال إلى الخلف)

- تطبيق ٩ : صمم برنامجاً لـForm يبقى دائماً في المقدمة بالنسبة للـForms وبالإمكان إحضار الـForm إلى الأمام أو إرساله إلى الخلف ؟
  - قم بإنشاء Project جديد .. تظهر نافذة الـForm1
  - قم بإضافة Form جديد بإسم Form2 من القائمة Project ــ..Add Windows Form>
  - عُد إلى الـForm1 وأضف أداة زر Button بإسم button1 وأضبط الخاصية Text له بالقيمة "في المقدمة" .
  - عُد إلى الـForm2 وأضف إليه ٢ أداة زر Button الأول بإسم button1 وأضبط الخاصية Text له بالقيمة "إحضار إلى الأمام" والثاني بإسم button2 وأضبط الخاصية Text له بالقيمة "إرسال إلى الخلف" .
  - سننتقل إلى نافذة الـCodeللـForm1 .. في قسم التصريحات إكتب الكود بالشكل :

```
    publicpartialclassForm1 : Form
        {
    Form2 f = newForm2();
```

  - في الحدث Clickللـbutton1 (في المقدمة) نكتب الكود بالشكل :

```
    privatevoid button1_Click(object sender, EventArgs e)
            {
                f.TopMost = true;
                f.Show();
            }
```

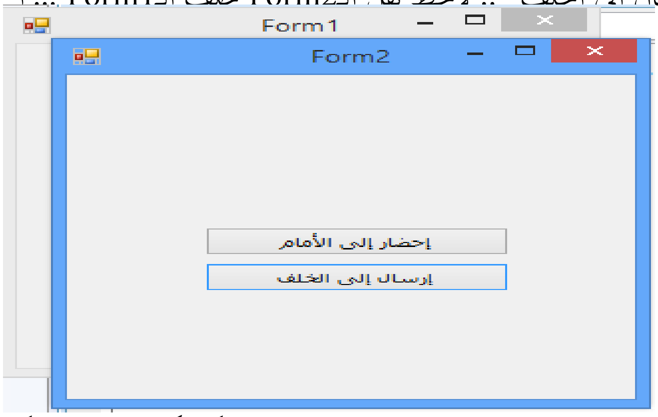
- ننتقل إلى نافذة الـCodeللForm2 .. في الحدث button1للClick نكتب الكود بالشكل :

```
privatevoid button1_Click(object sender, EventArgs e)
        {
this.BringToFront();
        }
```

- في الحدث button1للClick نكتب الكود بالشكل :

```
privatevoid button2_Click(object sender, EventArgs e)
        {
this.SendToBack();
        }
```

- قم بتنفيذ البرنامج .. إنقر على الزر "في المقدمة" .. لاحظ ظهور الـform2 في المقدمة .. إنقر على الزر "إرسال إلى الخلف" .. لاحظ نقل الـForm2 خلف الـForm1 ... إستمتع



// ملاحظة : بالنسبة لخصائص وطرق الـForm الحالي يمكن كتابتها داخل الكود مباشرة بدون كتابة كلمة this ... البرنامج يفهم تلقائياً أنّ هذه الخاصية أو الطريقة تتبع الـForm الحالي ..
مثلاً : قم بإنشاء Project جديد .. تظهر نافذة الـForm1 .. في الحدث LoadللForm1 إكتب الكود بالشكل :

```
privatevoid Form1_Load(object sender, EventArgs e)
        {
            CenterToScreen();
        }
```

- قم بتنفيذ البرنامج .. تظهر الـForm في وسط الشاشة .. لاحظ كتابة الطريقة CenterToScreen بدون كتابة كلمة this قبلها .. مباشرةً يفهم المترجم أنّ هذه الطريقة تتبع الـForm الحالي .

- تمارين غير محلولة :

١- صمم برنامجاً لـFormواحد مضاف إليها أدوات بمجرد تكبير حجم الـForm يتم تكبير حجم الأدوات الموجودة بداخله وعند تصغير حجم الـForm يتم تصغير حجم الأدوات الموجودة بداخله .

٢- صمم برنامجاً لـForm واحد مضاف إليه أدوات Controls متنوعة ... بمجرد النقر على الزر يتم عرض أسماء الأدوات المضافة كاملة (خاصية Name للأدوات) داخل أداة ListBox حسب ترتيب إضافتها .

٣- صمم برنامجاً لـForm واحد تتحرك على الشاشة بشكل مستمر من اليسار إلى اليمين كل ١٠٠ملي ثانية .. وعند وصولها إلى نهاية يمين الشاشة تظهر مرة أخرى من بداية يسار الشاشة (لنطلق على البرنامج الـForm المتحرك) .

٤- صمم برنامجاً لـForm1 يجعل النص الموجود في شريط العنوان للـForm يتحرك مثل الشريط الإخباري .

٥- صمم برنامجاً يحتوي على ٣نماذج Forms وقم بالتنقل بينForm1ـ> Form2ــ> Form3 ذهاباً وإياباً عن طريق النقر على أزرار button .

الفصل السادس

تطبيقات متنوعة على الـForm

Multiple Application on The Forms

- توجد تطبيقات أخرى كثيرة ومتعددة على الـForm .. في هذا الفصل سنستعرض أهم هذه العمليات وسيتم تصميم برامج توضحها مع الشرح …

- أولاً : إضافة الأدوات على الـForm وإزالتها منه برمجيا :

تعلمنا سابقاً كيف يتم إضافة أداة من الـToolbox إلى الـForm يدوياً في زمن التصميم (بالنقر المزدوج على الأداة أو بسحب الأداة وإفلاتها على الـForm) وكذلك كيف يتم إزالة الأداة من سطح الـForm (بالتحديد على الأداة ثم الضغط على المفتاح Delete) … لكن كيف يتم إضافة أداة إلى الـForm بعد تنفيذ البرنامج (في زمن التنفيذ) ويعني إضافة أي أداة أو حذفها برمجياً سواء كانت Button أو Label أو TextBox أو غيرها من الأدوات ويتم ضبط الخصائص أيضاً برمجياً ويتم ذلك بإستخدام الطريقتين Add , Remove من المجموعة Controls ..

- تطبيق ١٠ : في هذا المثال سيتم إضافة أداة مربع النص TextBox أو إزالتها برمجياً ؟

- قم بإضافة Project جديد بإسم P10 .. تظهر الـForm قم بإضافة زري Button واضبط الخصائص لهما كما في الجدول :-

الأداة	الخاصية Name	الخاصية Text
button1	btnadd	إضافة مربع نص
Button2	btnremove	إزالة مربع النص

- في قسم التصريحات أعلى شاشة الـCode نكتب الكود بالشكل :

```
public partial class Form1 : Form
    {
        TextBox t = new TextBox();
```

// قمنا هنا بإنشاء Object بإسم t من العائلة TextBox …

- في الحدث Click لـ btnadd نكتب الكود بالشكل :

```
private void btnadd_Click(object sender, EventArgs e)
        {
            t.Size = new Size(200,200);
            t.Location = new Point(30, 40);
            t.Font = new Font("Arial", 20f);
            t.BackColor = Color.Yellow;
            t.ForeColor = Color.Red;
            this.Controls.Add(t);
        }
```

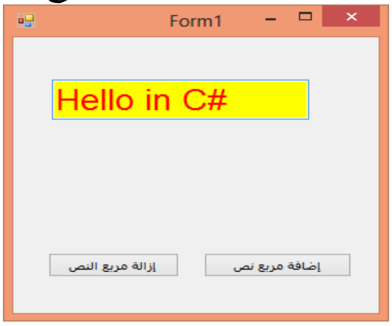
// في السطر الأول ضبطنا حجم مربع النص .. في السطر الثاني حددنا موقع مربع النص على الـForm .. في السطر الثالث حددنا نوع وحجم النص في مربع النص .. في السطر الرابع حددنا لون خلفية مربع النص .. في السطر الخامس حددنا لون الخط في مربع النص .. في السطر الأخير أضفنا مربع النص إلى الـForm الحالي .

- في الحدث Click لـ btnremove نكتب الكود بالشكل :

```
private void btnremove_Click(object sender, EventArgs e)
        {
            this.Controls.Remove(t);
       }
```

// قمنا بإزالة مربع النص من على الـForm بإستخدام الطريقة Remove …

- قم بتنفيذ البرنامج .. إنقر الزر "إضافة مربع نص " .. لاحظ إنشاء مربع النص .. إكتب فيه النص التالي "Hello in C#" .. قم بالنقر على الزر "إزالة مربع النص" .. لاحظ إزالته ...كما في الصورة



- ثانياً : تمرير الـForm إلى الإجرائيات :
يمكنك تمرير أي Object من الـForm إلى الإجرائيات (الدوال) بكل سهولة مثلها مثل أي أداة أخرى أو أي نوع Parameter آخر ...

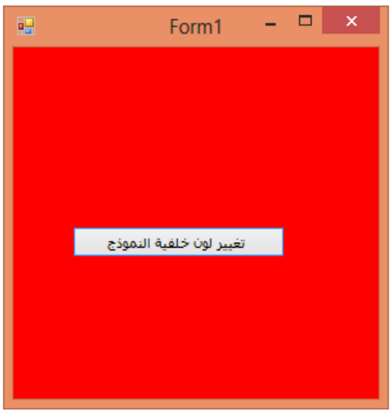
● تطبيق ١١: صمم برنامجاً يوضح كيفية تمرير الـForm إلى الإجرائيات ؟
- قم بإنشاء Project جديد بإسم P11 .. تظهر الـForm1 .. قم بإضافة زر Button ثم إضبط خاصية الـText له بالقيمة "تغيير لون خلفية النموذج" ...
- قم بإنشاء طريقة أو دالة بإسم ColorMyForm ذات بارامتر من نوع Form واكتب في نافذة الـCode الشيفرة التالية بالكامل كما في الشكل التالي :-

```
private void ColorMyForm(Form MyForm) {
            MyForm.BackColor = Color.Red;
        }
```

- في الحدث Click لـ button1 إكتب الكود بالشكل :

```
private void button1_Click(object sender, EventArgs e)
        {
            ColorMyForm(this);
        }
```

// قمنا بتمرير الـForm الحالي كبارامتر إلى الطريقة ColorMyForm ...
- قم بتنفيذ البرنامج .. إنقر على الزر "تغيير لون خلفية النموذج" .. لاحظ تغير لون خلفية الـForm الحالي كما في الإجرائية ColorMyForm ... كما في الصورة :



- ثالثاً : الرسم على الـForm :

قديماً كان الرسم بالحاسوب بواسطة أكواد برمجية بالصعوبة بمكان بسبب مشكلات تظهر لعدم تناسق عتاد وبرمجيات الرسوم، لكن شركة Microsoft واجهت هذه المشكلة من خلال تقديم طبقة من التجريد abstraction بين عتاد Hardware العرض وبين البرامج التي تستخدمه ، كانت هذه الطبقة جزءاً من Windows API يدعى واجهة الأجهزة الرسومية (Graphic Device Interface)GDI .

بوجود GDI سيقوم البرنامج برسم الرسوميات إلى ما يدعى بسياق الجهاز (Device Context) بدلاً من الكتابة مباشرة إلى بطاقة العرض . يقوم نظام التشغيل بعدها بأخذ هذا الرسم من سياق الجهاز ويحوله إلى بطاقة الفيديو أو الطابعة وبذلك يسمح لمبرمجي التطبيقات بالتركيز على منطق أعمالهم وليس على برامج تشغيل العتاد .

يشمل إستخدام GDI إنشاء مصادر نظام تشغيل محددة كالأقلام Pen والفرشاة Brush ثم إستخدامها لرسم الأشكال والنصوص ، يحتفظ نظام التشغيل بمقابض handle على هذه المصادر ولسوء الحظ فإنه يمتلك عدداً محدوداً من المقابض وبالتالي عندما تنتهي من إستخدام هذه المصادر فإن عليك أن تتذكر أن تحررها وإلا فإن نظام التشغيل سيستنفذ المقابض .

لم يكن التعامل مع GDI صعباً ولكن الطبقة الإضافية من التجريد قد تسببت ببعض البط ، ولهذا السبب قامت Microsoft في نهاية الأمر بإنشاء DirectX من أجل مطوري الألعاب ، حيث أن DirectX هو مجموعة من واجهات المستخدم التي تؤمن الوصول السريع إلى العتاد كما أنه يدعم التشكيل ثلاثي الأبعاد (3D rendering) الذي توفره الألعاب هذه الأيام أما بالنسبة لباقي تطبيقات Windows فإن GDI يعمل بشكل جيد .

تتضمن بيئة NET. نسخة مدارة (managed) من GDI تدعى +GDI وبوجودها يمكنك إنجاز رسوماتك بإستخدام كائن من الصنف Graphics . يمثل كائن Graphics سطح رسم كأن يكون نموذجاً أو مستند طابعة وهو يتضمن الكثير من المناهج من أجل رسم الأشكال والخطوط والنصوص وعرض الصور .

يتم وضع شيفرة الرسم عادة ضمن الحدث Paint للـ Form والذي يعني إعادة رسم النموذج ، يتلقى الحدث Paint وسيطاً من النوع PaintEventArgs الذي يتضمن كائن من Graphics من أجل الـForm الذي قام بقدح الحدث . يمكنك بعدها بإستخدام كائن Graphics هذا لإنجاز أية أعمال رسم ضرورية في التطبيق .

- رسم نص على الـ Form :

يمتلك الكائن Graphics الطريقة DrawString من أجل رسم نص يتم إستخدامها بتقديم سلسلة محرفية لعرضها ...وتستقبل هذه الطريقة ٤ بارامترات كائنات تمثل النص المراد رسمه والخط ولون الفرشاة وموقع الرسم على الـForm .

- تطبيق ١٢: صمم برنامجاً يقوم برسم إسمك على الـ Form ؟
  - قم بإنشاء Project جديد بإسم P12 .. تظهر الـ Form1 ...
  - في الحدث Paint لل Form1 إكتب الكود بالشكل التالي :

```
private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g=e.Graphics;
            string text="Mahmud";
            Font font=new Font("Arial",20f);
            Brush brush=Brushes.Blue;
            Point point = new Point(50, 100);
            g.DrawString(text, font, brush, point);

        }
```

// في السطر الأول أنشئنا Object من الصنف Graphics .. في السطر الثاني قمنا بتخزين النص المراد رسمه داخل متغير من نوع String .. في السطر الثالث حددنا نوع وحجم خط النص .. في السطر الرابع حددنا لون فرشاة الرسم .. في السطر الخامس حددنا موقع الرسم على الـForm

- قم بتنفيذ البرنامج .. لاحظ رسم النص Mahmud على الـ Form كما في الصورة :



- رسم خطوط مستقيمة على الـ Form :

يمكن رسم خط مستقيم على الـForm بإستخدام الطريقة DrowLine التي تتبع الكائن Graphics وتستقبل هذه الطريقة ٣بارامترات كائنات تمثل قلم الرسم Pen و موقع بداية الرسم ونهايته .

- تطبيق ١٣ : صمم برنامجاً لرسم ١٠ خطوط مستقيمة على الـForm بلون قرمزي وبطول متزايد؟
  - قم بإنشاء Project جديد .. تظهر الـ Form1 ...
  - في الحدث Paint لل Form1 قم بكتابة الكود بالشكل التالي :

```
private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            Pen pen = Pens.Crimson;
            Point start = new Point(20, 50);
            Point stop = new Point(20, 100);
            for (int i = 0; i < 10; i++)
            {
                start.X += 20;
                stop.X += 20;
                stop.Y += 20;
                g.DrawLine(pen, start, stop);
            }
                }
```

// في السطر الأول عرفنا Object من الصنف Graphics .. في السطر الثاني حددنا لون القلم قرمزي .. في السطر الثالث حددنا نقطة بداية الرسم لأول خط .. في السطر الرابع حددنا نقطة نهاية الرسم لأول خط .. حلقه For لرسم ١٠خطوط بحيث يبعد كل خط عن الآخر ٢٠نقطة ويزيد كل طول خط عن الآخر ٢٠نقطة أعلى وأسفل ...

- قم بتنفيذ البرنامج ... لاحظ رسم ١٠خطوط مستقيمة بشكل رائع كما في الصورة :



- رسم الأشكال الهندسية (دائرة – مستطيل - ...) على الـ Form :-
- رسم الأشكال البيضاوية :

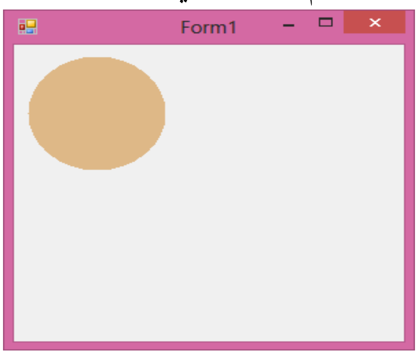
يقدم الصنف Graphics الطريقة FillEllipse من أجل رسم الأشكال البيضاوية ولكي تتمكن من إستخدام FillEllipse عليك تقديم مستطيل للمنطقة المحيطة بالشكل البيضاوي ويقوم كائن Graphics بإنجاز الرسم الفعلي فإذا تصادف أن كان المستطيل المحيطي الذي قدمته هو مربع تماماً فإن الشكل البيضاوي سيكون دائرة ١.

- تطبيق ١٤: صمم برنامجاً لرسم دائرة بالفرشاة وبإستخدام اللون خشبي على الـForm ؟
  - قم بإنشاء Project جديد بإسم P14.. تظهر الـForm1 ...
  - في الحدث Paint للـForm1 قم بكتابة الكود بالشكل التالي :

```
private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            SolidBrush azurbrush = new SolidBrush(Color.BurlyWood);
            Rectangle circlebound = new Rectangle(10, 10, 100, 100);
            g.FillEllipse(azurbrush, circlebound);
        }
```

// في السطر الأول أنشئنا Object من الصنف Graphics .. في السطر الثاني حددنا لون فرشاة رسم الشكل .. في السطر الثالث حددنا أبعاد المستطيل الذي سترسم الدائرة ضمن حدوده .

- قم بتنفيذ البرنامج .. لاحظ رسم الدائرة كما في الصورة :



- رسم الأشكال الرباعية :

يقدم الصنف Graphics الطريقة FillRectangle لرسم الأشكال الهندسية ذات أربعة أضلاع مثل المربع والمستطيل ... بحيث تقوم بملأ المنطقة المستطيلة الموجودة ضمن مستطيل حدودي معطى كبارامتر ثاني للطريقة بالإضافة إلى كائن يمثل لون فرشاة الرسم ...

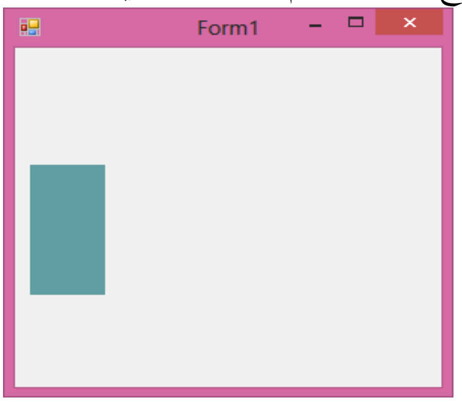
- تطبيق ١٥: صمم برنامجاً لرسم مستطيل على الـForm بلون أزرق فاتح ؟
  - قم بإنشاء Project جديد بإسم P15 .. تظهر الـ Form1 ...
  - في الحدث Paint لـ Form1 نكتب الكود بالشكل التالي :

```
private void Form1_Paint(object sender, PaintEventArgs e)

        {
            Graphics g=e.Graphics;
            SolidBrush beigebrush = new SolidBrush(Color.CadetBlue);
            Rectangle rectbound = new Rectangle(10, 90, 50, 100);
            g.FillRectangle(beigebrush, rectbound);
        }
```

// في السطر الأول أنشئنا Object من الصنف Graphics .. في السطر الثاني حددنا لون فرشاة رسم الشكل .. في السطر الثالث حددنا أبعاد المستطيل ...

- قم بتنفيذ البرنامج .. لاحظ رسم مستطيل كما في الصورة :



- تفريغ المصادر:

تتضمن كائنات الرسوميات كالفرشاة Brush والأقلام Pen مصادر نظام تشغيل غير مدارة unmanaged لن يعرف جامع نفايات NET. كيف يحررها ، يتم عادة تحقيق الواجهة IDispose من قبل الأصناف التي تتضمن مصادر غير مدارة وتحتاج إلى أن يتم تحريرها بشكل صريح .
حيث الواجهة تصرح عن الطريقة Dispose الذي يستخدمه تحقيق الأصناف لتحرير المصادر غير المدارة كقابض Pen ، Brush . يقوم السطرين التاليين من معالج Paint بالنسبة للبرنامجين الأخيرين بإستدعاء الطريقة Dispose على الفرشاتين وذلك بهدف تفريغ المصادر :

```
azurebrush.Dispose();
beigebrush.Dispose();
```

قد يبدو أن معالج الحدث Paint السابق ينجز العمل بشكل جيد بإستدعائه للطريقة Dispose ، لكن مازالت هناك إمكانية أن يتم إلقاء إستثناء مما يعني الخروج من الطريقة Form1_Paint بطريقة غير متوقعة قبل إستدعاء Dispose . فإذا حدث هذا فإن البرنامج سيعاني من تسرب

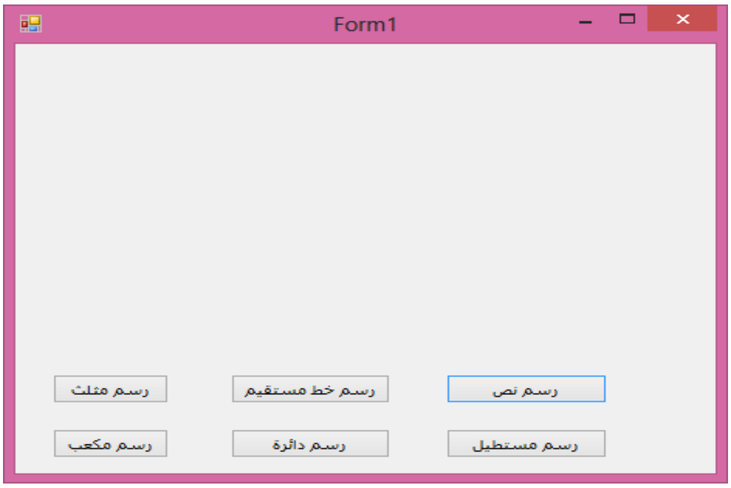
المصادر . وللحماية من تسرب المصادر تقدم #C طريقة لتعريف كتلة من الشيفرة تضمن أن يتم إستدعاء الطريقة Dispose .

تقوم الكتلة using بإنشاء قسم محمي من الشيفرة في مكان توضع المصادر ، وعند الخروج من هذه الكتلة سيتم إستدعاء الطريقة Dispose أوتوماتيكياً على الكائنات التي تم إنشاؤها في أعلى الشيفرة كالتالي :

```
using( SolidBrush beigebrush = new SolidBrush(Color.CadetBlue);)
```

- تمارين غير محلولة :

١- صمم برنامجاً يقوم بالرسم على الـForm بمجرد النقر على الزر كما في الواجهة التالية :



٢- صمم برنامجاً يقوم بإنشاء زر button برمجياً عند إقلاع الـForm وعند النقر على زر الـbutton تظهر رسالة "Hello in C#" ؟

٣- صمم برنامجاً يوجد به 2 Forms بحيث يتم تمرير الـForm2 إلى الإجرائيات داخل الـForm1 بحيث يقوم بتغيير لون خلفية الـForm2 داخل هذه الإجرائية ويتم إستدعاء الإجرائية عند النقر على زر button موجود على الـForm1 ؟

٤- كيف يمكن رسم زر الـbutton بحيث يظهر بشكل دائري ؟

٥- صمم برنامجاً لحساب مساحة وحجم الأشكال الهندسية مع رسمها وفق الأبعاد المدخلة حسب كل شكل هندسي بحيث يشمل مختلف الأشكال الهندسية الشائعة الإستخدام :

مربع ، مستطيل ، مثلث ، متوازي أضلاع ، دائرة ، قطع ناقص ، ... إلخ .

الفصل السابع

التعامل مع صندوق الرسائل

Working With MessageBox

- للرسائل أهمية كبيرة في البرمجة بأنواعها فهي تساعد على إدارة البرامج وتخبر المستخدم بمعلومات عن النظام وعن حالة العمليات فيه وتعتبر واجهة الحوار الأكثر شيوعاً والتي يستخدمها البرنامج للتحكم بأفعال المستخدم وتوجيهه وتأتي في مكتبات NET. بعدة أشكال وأنواع ... ويتم عرضها داخل صندوق يسمى صندوق الرسائل MessageBox وهي لا تعتبر أداة في VC# وإنما يعتبر Class يمتلك ٣طرق فقط أهمها الطريقة Show والتي تستخدم لعرض الـMessageBox كما سيتم شرحها لاحقاً ...كما يمكن عرض الرسائل في أي حدث من الأحداث ولأي أداة من الأدوات وفي أي وقت من أوقات تشغيل البرنامج في زمن التنفيذ ويتم إعداد الرسائل في زمن البرمجة ...
- يمتلك الصنف MessageBox الطريقة Show والتي تستخدم لعرض الرسائل وتظهر الطريقة Show بما يقارب ٢١شكل تقريباً وتحتوي على عدة بارامترات يفصل بين كل بارامتر وآخر فاصلة , يمكن شرح أهم هذه البارامترات كالتالي :

١- string text : ويمثل نص الرسالة ويكتب محتوى الرسالة داخل علامتي تنصيص مزدوجة " " ويظهر محتوى الرسالة في وسط صندوق الرسالة MessageBox .

٢- string caption : ويمثل عنوان الرسالة ويكتب أيضاً داخل علامتي تنصيص مزدوجة " " ويظهر عنوان الرسالة في شريط عنوان صندوق الرسالة في الأعلى .

٣- MessageBoxButtons buttons : ويمثل أزرار الرسالة التي تظهر في صندوق الرسائل أسفل نص الرسالة وتوجد عدة أنواع من الأزرار كالتالي :
- `AbortRetryIgnore` : وتظهر ٣ أزرار في الرسالة وهي Abort (إجهاض الأمر) ، Retry (إعادة المحاولة) ، Ignore (تجاهل) .
- `OK` : يظهر زر واحد فقط هو OK (موافق) .
- `OKCancel` : يظهر زرين هما OK (موافق) ، Cancel (إلغاء الأمر) .
- `RetryCancel` : يظهر زرين هما Retry (إعادة المحاولة) ، Cancel (إلغاء الأمر) .
- `YesNo` : يظهر زرين هما Yes (نعم) ، No (لا) .
- `YesNoCancel` : تظهر ٣ أزرار هي Yes (نعم) ، No (لا) ، Cancel (إلغاء الأمر) .

٤- MessageBoxIcon icon : وتمثل أيقونة الرسالة وتعتبر شكل أيقونة الرسالة دال على نوعها وتظهر الأيقونة بجانب نص الرسالة وتوجد الأيقونة على عدة أشكال كالتالي :
- `Asterisk` : أيقونة تعرض حرف i ضمن دائرة زرقاء اللون .

- Error : أيقونة تعرض حرف x باللون الأبيض في دائرة حمراء اللون .
- Exclamation : أيقونة تعرض إشارة تعجب ضمن مثلث أصفر اللون .
- Hand : أيقونة تعرض حرف x باللون الأبيض في دائرة حمراء اللون .
- Information : أيقونة تعرض حرف i ضمن دائرة زرقاء اللون .
- None : بدون أية أيقونة .
- Question : أيقونة تعرض إشارة إستفهام في دائرة .
- Stop : أيقونة تعرض حرف x باللون الأبيض في دائرة حمراء اللون .
- Warning : أيقونة تعرض إشارة تعجب ضمن مثلث أصفر اللون .

٥- MessageBoxDefultButton defultButton :وتمثل الزر الإفتراضي الذي يقع عليه التركيز حال ظهور الرسالة وتوجد ٣ خيارات كالتالي :

- Button1: تجعل الزر الأول في مربع الرسالة الزر الإفتراضي .
- Button2 : تجعل الزر الثاني في مربع الرسالة الزر الإفتراضي .
- Button3 : تجعل الزر الثالث في مربع الرسالة الزر الإفتراضي .

٦- MessageBoxOptions options : وتمثل خيارات عرض الرسالة وتوجد ٤ خيارات كما يلي :

- DefaultDesktopOnly : تعرض الرسالة على سطح المكتب الفعّال .
- RightAlign : تكون محاذاة النص في مربع الرسالة إلى اليمين .
- RtlReading : تحدد أن ترتيب قراءة النص المعروض من اليمين إلى اليسار .
- ServiceNotification : تظهر الرسالة على شكل إشعارات الخدمة .

- تنتمي نتيجة الطريقة Show إلى مجموعة نمط التعداد DialogResult وهي تدل على الزر الذي نقر عليه المستخدم وتحتوي هذا النمط على عدة خيارات (قيم) كالتالي :
  - Abort : يدل على أن الزر الذي نقر عليه المستخدم هو Abort .
  - Cancel : يدل على أن الزر الذي نقر عليه المستخدم هو Cancel .
  - Ignore : يدل على أن الزر الذي نقر عليه المستخدم هو Ignore .
  - No : يدل على أن الزر الذي نقر عليه المستخدم هو No .
  - None : لم ينقر المستخدم على أي زر أي أن مربع الحوار الشرطي مستمر بالعمل .
  - Ok : يدل على أن الزر الذي نقر عليه المستخدم هو Ok .
  - Retry : يدل على أن الزر الذي نقر عليه المستخدم هو Retry .
  - Yes : يدل على أن الزر الذي نقر عليه المستخدم هو Yes .

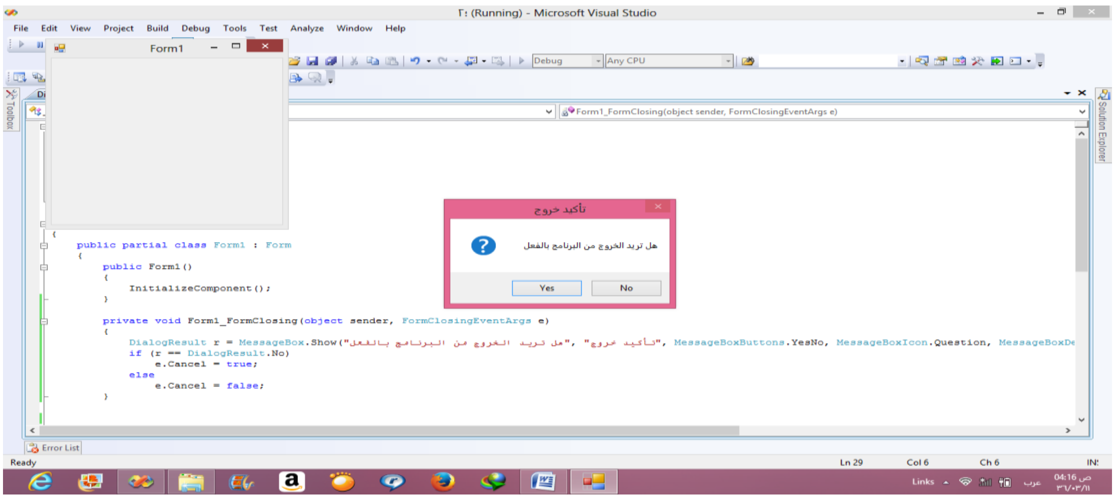
- تطبيق ١٦: صمم برنامجاً يظهر للمستخدم رسالة تأكيد خروج عند النقر على الزر X في شريط عنوان الـForm .. في حال تم النقر على الزر Yes يتم الخروج من البرنامج أما إذا تم النقر على الزر No لا يتم إغلاق الـForm وبالتالي عدم الخروج من البرنامج ؟
  - أنشأ Project جديد بإسم P16 .. تظهر الـ Form1 ...
  - في الحدث FormClosing لـ Form1 إكتب الكود بالشكل التالي :

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
DialogResult r = MessageBox.Show(" هل تـريـد الخـروج مـن الـبـرنـامـج بـالـفـعـل ",
"تـأكـيـد خـروج ", MessageBoxButtons.YesNo, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button1, MessageBoxOptions.RightAlign);

 if (r == DialogResult.No)
                e.Cancel = true;
            else
                e.Cancel = false;
        }
```

// السطر ;e.cancel=True يقوم بتعطيل حدث الإغلاق .

  - قم بتنفيذ البرنامج .. إنقر على الزر X تظهر رسالة تأكيد الخروج إنقر الزر No لاحظ عدم الخروج من البرنامج .. أعد النقر مرة أخرى على الزر X تظهر رسالة تأكيد الخروج إنقر الزر Yes يتم الخروج من البرنامج .



- ظهور أكثر من رسالة في برنامج واحد :
  يمكن إظهار أكثر من رسالة في نفس الـProject أو في نفس الـForm من المشروع بحيث يتم إستدعاء الطريقة Show من الصنف MessageBox في كل مرة أردنا فيها إظهار رسالة ويمكن أن تظهر كل رسالة بشكل مختلف عن الأخرى من حيث أزرار الرسالة وأيقونة الرسالة ...

- تطبيق ١٧: صمم برنامجاً لمسابقة عواصم الدول يتكون من ٣ أسئلة كل سؤال يظهر في رسالة ويكون الجواب عليها بنعم أو لا ودرجة كل سؤال ١٠ درجات وفي نهاية المسابقة تظهر رسالة فيها درجة المتسابق ؟

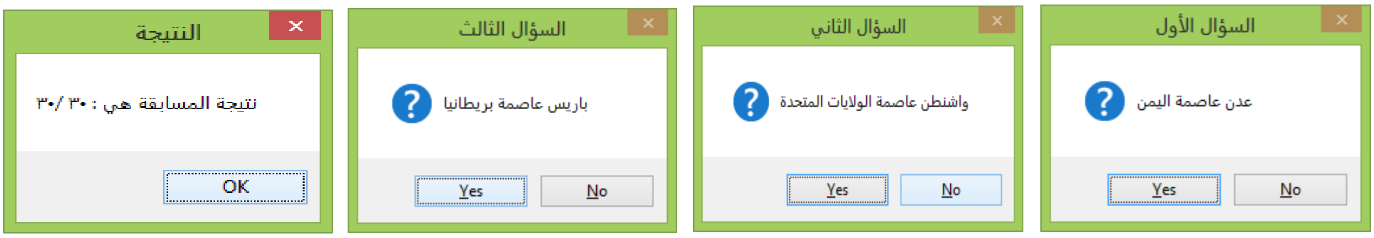
- أنشأ Project جديد بإسم P17 .. تظهر الـ Form1 ...

- في الحدث Load لـ Form1 إكتب الكود بالشكل التالي :

```
private void Form1_Load(object sender, EventArgs e)
        {
            int deg = 0;
            DialogResult r1, r2, r3;
            r1 = MessageBox.Show("عدن عاصمة اليمن  ", "السؤال الأول  ",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (r1 == DialogResult.No)
                deg += 10;
  r2 = MessageBox.Show(" واشنطن عاصمة الولايات المتحدة ", "السؤال الثاني ",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (r2 == DialogResult.Yes)
                deg += 10;
            r3 = MessageBox.Show("باريس عاصمة بريطانيا ", "السؤال الثالث ",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (r3 == DialogResult.No)
                deg += 10;
           MessageBox.Show("نتيجة المسابقة هي : " + deg.ToString() + " /30"
, "النتيجة");
            this.Close();

        }
```

- نفذ البرنامج وأجب بنعم أو لا ثم لاحظ النتيجة .. كما في الصورة :



الفصل الثامن

أحداث الفأرة وأحداث لوحة المفاتيح

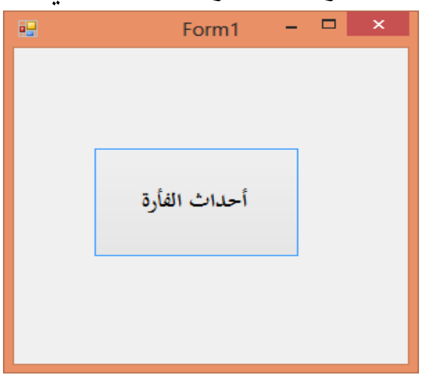
Mouse Events & Keyboard Events

- تشترك الكثير من أدوات التحكم Controls والـForm في أنها تمتلك نفس أحداث الفأرة وأحداث لوحة المفاتيح وهذه الأحداث تتولد من خلال أفعال المستخدم التي يحدثها على الفأرة ولوحة المفاتيح .
- أدوات التحكم Controls : هي الأدوات التي تضاف على سطح الـForm أما الأدوات الأخرى تضاف أسفل الـForm وأشهر أدوات الـ Controls : مربعات النصوص مثل TextBox ،...– الأزرار مثل Button ،... – الملصقات مثل Label ، ... – القوائم مثل ListBox ، ... –... إلخ.
- أحداث الفأرة Mouse Events :

نستطيع معالجة الاحداث المتولدة عن إستخدام الفأرة ، حركة الفأرة في الـForm والـControls وكتابة الأكواد البرمجية بداخلها .. سنورد فيما يلي أحداث الفأرة الممكنة في الصنف Control الذي يشكل الأساس للنماذج وعناصر التحكم كالتالي :

- MouseCaptureChanged : يقع عند تغير لقطة مؤشر الفأرة .
- MouseClick : يقع عند النقر على عنصر التحكم نقرة واحدة .
- MouseDoubleClick : يقع عند النقر نقراً مزدوجاً على عنصر الفأرة ويوجد هذا الحدث للـForm فقط .
- MouseDown : يقع عندما يضغط المستخدم زر الفأرة نحو الأسفل بينما يكون مؤشر الفأرة فوق عنصر التحكم .
- MouseUp : يقع عندما يحرر المستخدم الفأرة (بعد الضغط على زر الفأرة ورفع الإصبع عنه) بينما يكون مؤشر الفأرة فوق عنصر التحكم .
- MouseEnter : يقع عندما يدخل مؤشر الفأرة حدود عنصر التحكم .
- MouseHover : يقع عندما يحوم مؤشر الفأرة على عنصر التحكم (أي يقف فترة من الزمن) .
- MouseLeave : يقع عندما يغادر مؤشر الفأرة حدود عنصر التحكم .
- MouseMove : يقع عندما يتحرك مؤشر الفأرة فوق عنصر التحكم .

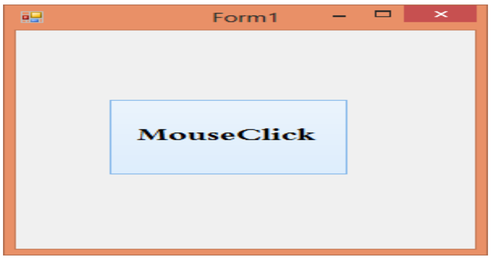
- تطبيق ١٨ : صمم برنامجاً توضح فيه الفرق بين أحداث الفأرة وكيفية حدوثها ؟
  - أنشأ Project جديد بإسم P18 .. تظهر الـForm1 .. أضف زر Button ثم إضبط الخاصية Text له بالقيمة "أحداث الفأرة" ... تظهر الـForm كما في الصورة :



- في الحدث MouseClick للـbutton1 نكتب الكود بالشكل :

```
private void button1_MouseClick(object sender, MouseEventArgs e)
        {
            button1.Text = "MouseClick";
        }
```

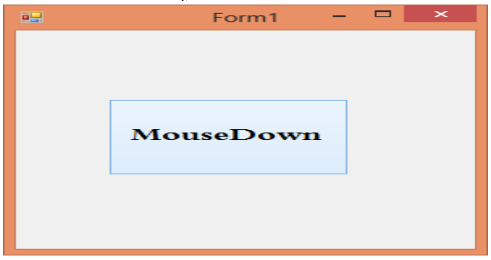
- نفذ البرنامج .. قم بالنقر على زر الـButton .. لاحظ تغير عنوان الـButton



- في الحدث MouseDown للـbutton1 إكتب الكود بالشكل :

```
private void button1_MouseDown(object sender, MouseEventArgs e)
        {
            button1.Text = "MouseDown";
        }
```

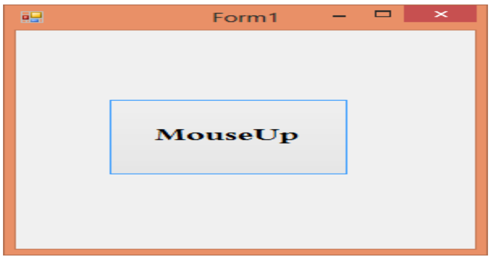
- نفذ البرنامج .. حرك مؤشر الفأرة فوق زر الـButton ثم إضغط زر الفأرة نحو الأسفل بدون أن ترفع إصبعك عنها .. لاحظ تغير عنوان الـButton كما في الصورة :



- في الحدث MouseUp للـbutton1 إكتب الكود بالشكل :

```
private void button1_MouseUp(object sender, MouseEventArgs e)
        {
            button1.Text = "MouseUp";
        }
```

- نفذ البرنامج .. حرك مؤشر الفأرة فوق زر الـButton ثم إضغط زر الفأرة ثم حرر الزر برفع الإصبع عنه .. لاحظ تغير عنوان الـButton كما في الصورة :

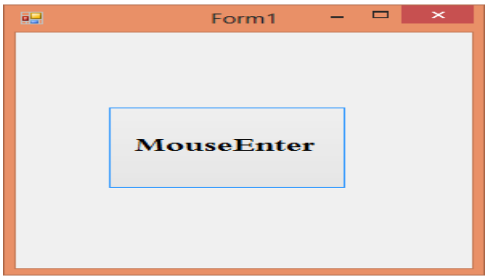


- في الحدث MouseEnter إكتب الكود بالشكل :

```
private void button1_MouseEnter(object sender, EventArgs e)
```

```
{
    button1.Text = "MouseEnter";
}
```

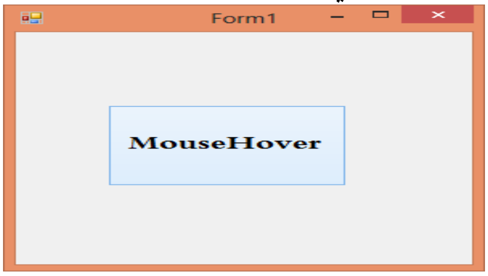
- نفذ البرنامج .. حرك مؤشر الفأرة إلى حدود زر الـButton .. لاحظ تغير عنوان الـButton



- في الحدث MouseHover للـbutton1 إكتب الكود بالشكل :

```
private void button1_MouseHover(object sender, EventArgs e)
{
    button1.Text = "MouseHover";
}
```

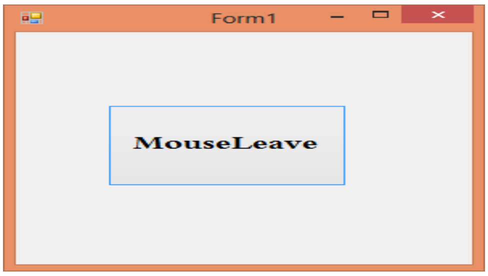
- نفذ البرنامج .. حرك مؤشر الفأرة فوق زر الـButton تماماً ثم توقف عن الحركة لفترة زمنية (يحوم) .. لاحظ تغير عنوان الـButton كما في الصورة :



- في الحدث MouseLeave للـbutton1 إكتب الكود بالشكل :

```
private void button1_MouseLeave(object sender, EventArgs e)
{
    button1.Text = "MouseLeave";
}
```

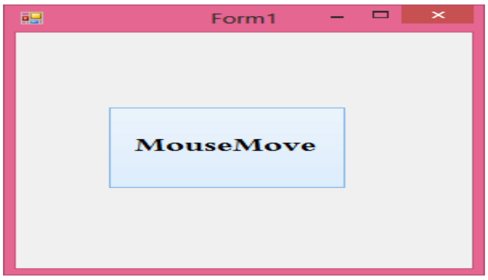
- نفذ البرنامج .. حرك مؤشر الفأرة فوق زر الـButton ثم غادر بمؤشر الفأرة خارج حدود زر الـButton .. لاحظ تغير عنوان الـButton كما في الصورة :



- في الحدث MouseMove للـbutton1 إكتب الكود بالشكل :

```
private void button1_MouseMove(object sender, MouseEventArgs e)
{
    button1.Text = "MouseMove";
}
```

- نفذ البرنامج .. حرك مؤشر الفأرة فوق زر الـButton بشكل مستمر وبدون توقف .. لاحظ الصورة :



- الكائن e MouseEventArgs يكون بارامتر في الطريقة أو المنهج (معالج الأحداث) لأحداث الفأرة التالية :

MouseMove & MouseUp & MouseDown & MouseClick فقط ، ويمتلك هذا الكائن عدة خصائص تظهر بعد كتابة (.e) وهي كالتالي :

- Button : تدل على أي زر من أزرار الفأرة قد ضغط عليه .
- Clicks : تدل على عدد المرات التي تم فيها ضغط زر الفأرة وتحريره .
- Delta : تدل على مجموع موجب أو سالب لعدد وحدات دورات عجلة الفأرة ، وتقدر وحدة الدوران بمقدار دوران عجلة الفأرة لمسافة تعادل 1 Notch .
- Location : تدل على قيمة الإحداثي (السيني والصادي) لموقع نقرة الفأرة .
- X : تدل على الإحداثية الأفقية (محور السينات) لنقرة الفأرة .
- Y : تدل على الإحداثية العمودية (محور الصادات) لنقرة الفأرة .

- تطبيق ١٩: صمم برنامجاً يقوم عند النقر على الـForm بمعرفة مايلي :

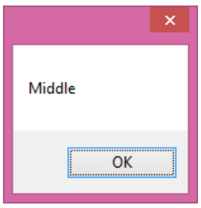
أ- زر الفأرة الذي تم النقر عليه من قبل المستخدم (الأيمن أو الأيسر أو الأوسط) ؟

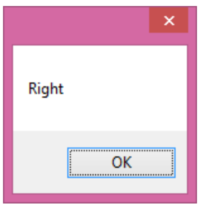
ب- موقع الإحداثي الأفقي والعمودي للمكان الذي تم النقر عليه من قبل المستخدم ؟

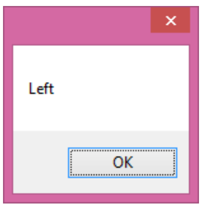
- أنشأ Project جديد بإسم P19 .. تظهر الـForm1 ...
- في الحدث MouseDown للForm1 نكتب الكود التالي :

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            MessageBox.Show(e.Button.ToString());
        }
```

- قم بتنفيذ البرنامج .. إنقر بالزر الأيمن للفأرة على الـForm تظهر رسالة "Right" .. إنقر بالزر الأيسر للفأرة تظهر رسالة "Left" .. إنقر على عجلة الفأرة (الزر الأوسط) تظهر رسالة "Middle" .. لاحظ الصور :







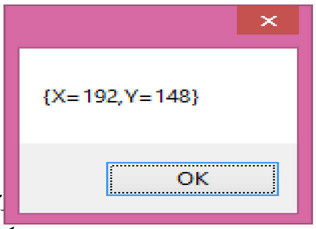
- عطل الكود السابق بـComment أو قم بمسحه .. في نفس الحدث MouseDown للـForm1 إكتب الكود بالشكل :

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
        {
            MessageBox.Show(e.Location.ToString());
        }
```

- أو يمكن كتابة الكود بالشكل :

```
MessageBox.Show(e.X.ToString()+ " , " + e.Y.ToString());
```

- نفذ البرنامج .. ثم إنقر في أي مكان على الـForm تظهر رسالة موقع الإحداثي كما في الصورة :



- أحداث لوحة المفاتيح Keyboard Events :

الأفعال التي تتولد بسبب تعامل المستخدم مع أي مفتاح من حة المفاتيح ، يمكن أن تعالج أحداث لوحة المفاتيح في النماذج Forms والعديد من عناصر التحكم Controls وذلك من خلال الأحداث التالية :

- KeyDown : يقع عندما يضغط المستخدم أحد المفاتيح للأسفل بينما يمتلك عنصر التحكم التركيز .
- KeyPress : يقع عندما يضغط المستخدم أحد المفاتيح بينما يمتلك عنصر التحكم التركيز .
- KeyUp : يقع عندما يحرر أحد المفاتيح (رفع الأصبع عن المفتاح بعد الضغط عليه) بينما يمتلك عنصر التحكم التركيز.

- تمتلك الطريقة (معالج الأحداث) من أجل الحدثين KeyUp & KeyDown بارامتر (وسيط) e من النوع KeyEventArgs والذي يمتلك عدة خصائص كالتالي :

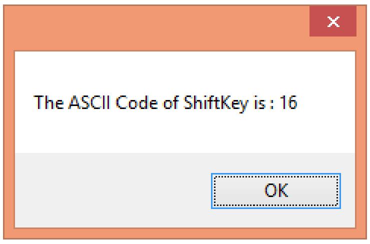
- Alt : تحوي قيمة من نوع Boolean فيما إذا كان المفتاح Alt قد ضُغط من قبل المستخدم عندها تكون القيمة True أما إذا ضُغط مفتاح آخر تكون القيمة False .
- Control : تحوي قيمة من نوع Boolean فيما إذا كان المفتاح Ctrl قد ضُغط من قبل المستخدم عندها تكون القيمة True أما إذا ضُغط مفتاح آخر تكون القيمة False .
- Shift : تحوي قيمة من نوع Boolean فيما إذا كان المفتاح Shift قد ضُغط من قبل المستخدم عندها تكون القيمة True أما إذا ضُغط مفتاح آخر تكون القيمة False .
- Handled : تحوي أو تضبط قيمة من نوع Boolean تحدد فيما إذا كان الحدث قد عُولج أم لا.
- KeyCode : تحوي ترميز لوحة المفاتيح للمفتاح الذي سبب الحدث KeyDown أو الحدث KeyUp .
- KeyData : تحوي بيانات لوحة المفاتيح للمفتاح من أجل الحدث KeyDown أو الحدث KeyUp .

- KeyValue : تحوي قيمة المفتاح بشيفرة الأسيكي ASCII Code بالنظام العشري من أجل الحدث KeyDown أو الحدث KeyUp مثلاً "A" قيمته 65 ... .
- Modifiers : تحوي على مؤشرات المحددات من أجل الحدث KeyDown أو الحدث KeyUp وهي تدل على أي من أزرار المحددات (Ctrl أو Shift أو Alt) قد ضُغط وفي هذه الحالة تكون القيمة Ctrl أو Shift أو Alt على التوالي كلاً على حده ويمكن أن تكون ناتج عملية OR على قيم المحددات ، أما إذا ضُغط على مفتاح آخر فإن القيمة تكون None .. ومن الطبيعي أن يكون إستخدام الخصائص Control & Shift & Alt أسهل من الخاصية Modifiers .

● تطبيق ٢٠ : صمم برنامجاً يقوم بمجرد الضغط على المفتاح يظهر رسالة تحتوي على ترميز المفتاح وقيمته بالنظام العشري في ASCII Code ؟
- أنشأ Project جديد بإسم P20 .. تظهر الـForm1 ...
- في الحدث KeyDown للـForm1 نكتب الكود بالشكل :

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
     MessageBox.Show("The ASCII Code of " + e.KeyCode.ToString() + " is : "
+ e.KeyValue.ToString());
        }
```

- نفذ البرنامج .. إضغط مثلاً على المفتاح Shift تظهر الرسالة كما في الصورة :

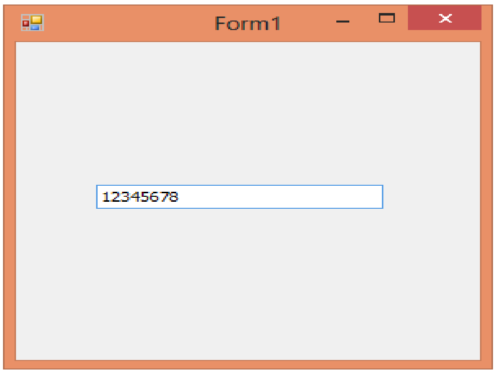


● تمتلك الطريقة (معالج الأحداث) من أجل الحدث KeyPress بارامتر (وسيط) e من النوع KeyPressEventArgs والذي يمتلك الخصائص التالية :
- Handled : تضبط أو تعيد قيمة من نوع Boolean تحدد فيا إذا كان الحدث KeyPress قد عولج أم لا ، فإذا ضُبطت هذه الخاصية على القيمة True فإن VC# لن تعالج هذا المفتاح ، وبذلك إذا أردت أن تحذفه فما عليك إلا أن تضبط الخاصية Handled على القيمة True دون الحاجة لأية معالجة إضافية .
- KeyChar : تحوي أو تعيد قيمة من نوع Char تمثل الحرف الموافق للمفتاح المضغوط .

● تطبيق ٢١: صمم برنامجاً لـForm تحتوي على مربع نص TextBox لا يقبل إلا أرقام ؟
- أنشأ Project جديد بإسم P21 .. تظهر الـForm1 .. أضف أداة TextBox إلى الـForm
- في الحدث KeyPress للـtextBox1 إكتب الكود بالشكل التالي :

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar < '0' || e.KeyChar > '9')
                e.Handled = true;
```

- قم بتنفيذ البرنامج .. أدخل أرقام إلى مربع النص .. لاحظ أنه يكتبها .. قم بإدخال حروف أو رموز .. لاحظ أنه لا يكتبها بمعنى أنه يعطل مفاتيح تلك الحروف أو الرموز .. كما في الصورة :



- تمارين غير محلولة :-

١- صمم برنامجاً لتعليم نطق الحروف الإنجليزية بحيث إذا قام المستخدم بالضغط على الحرف من لوحة المفاتيح فإن البرنامج يصدر صوتاً ينطق هذا الحرف ؟

٢- صمم برنامجاً لـForm يحتوي على مربع نص TextBox لا يقبل إلا حروف فقط مع إمكانية إستخدام مفاتيح التحرير التالية :-
مفتاح المسطرة Space – مفتاح الحذف Delete – مفتاح الحذف للخلف BackSpace ؟

٣- صمم برنامجاً لـForm يحتوي على زر Button بمجرد الضغط على زر الفأرة الأيسر فوق زر الـButton وسحبه من مكانه وإفلاته إلى مكان آخر على الـForm فإنه ينتقل إلى المكان الجديد (بمعنى تغيير موقع زر الـButton على الـForm بالسحب والإفلات في زمن التنفيذ) ؟

٤- صمم برنامجاً لـForm يحتوي على ٣ أدوات مربعات نصوص TextBox (تحت بعض) بحيث يتم التنقل بين مربعات النص بعد إدخال البيانات إليها بالضغط على المفتاح Enter (بدلاً عن المفتاح Tab) ؟

الفصل التاسع

الخصائص المشتركة بين الأدوات

Common properties between tools

- توجد مجموعة من الخصائص التي تشترك فيها أكثر من أداة Tool أو عنصر تحكم Control مثل الـTextBox ، Button ، Label ، ListBox ، ... إلخ وكذلك الـForm ، ويمكن ضبط هذه الخصائص يدوياً في زمن التصميم من نافذة الخصائص أو يمكن ضبطها برمجياً في زمن البرمجة بكتابة الأكواد البرمجة في نافذة الـCode .. سنسرد الخصائص مع بعض الأمثلة كالتالي :-

١- الخاصية Name : وتعني الإسم البرمجي للأداة وتستقبل وتعيد قيمة من نوع String وهي خاصية تمتلكها جميع الأدوات في VC#.net.. عند إضافة الأداة يتم إعطاء تسمية إفتراضية للأداة تضم إسم الأداة متبوعاً برقم مثلاً Form1 , textBox2، ... لكن على المبرمج المحترف إعادة تسمية الأداة بإستخدام البادئة (وتكون البادئة من ٣ أحرف) ملحوقة بكلمة تدل على وظيفة الأداة في البرنامج الحالي ويمكن سرد أسماء بعض الأدوات كما في الجدول :

الأداة	الإسم الإفتراضي	البادئة	مثال على تسمية برمجية
النموذج (Form)	Form1	frm	frmfrist
زر أمر (Button)	button1	btn	btnexit
مربع نص (TextBox)	textBox1	txt	txtnumber
أداة العنوان (ملصق) (Label)	label	lbl	lbl
مربع قائمة (ListBox)	listBox1	lst	lstoddnumber
مربع إختيار (CheckedBox)	checkedBox	chk	chktest
... وهكذا (سيتم كتابة التسمية البرمجية لكل أداة عند تناولها كل منها على حده)			

٢- الخاصية Text: وتعني عنوان الأداة أو الإسم الذي يظهر على الأداة .. تستقبل وتعيد قيمة من نوع String وتشترك فيها جميع عناصر التحكم مثل الـForm ، button ، Label ، ... ويتم إسناد قيمة نصية لها في جميع الأدوات في زمن التصميم والبرمجة فقط ما عدا أداة مربع النصTextBox ومشتقاتها فبالإمكان إسناد قيمة نصية في جميع الأزمنة بما فيها زمن التنفيذ .

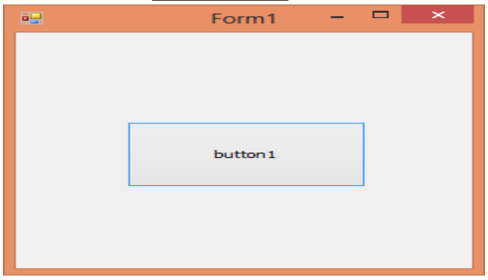
- تطبيق٢٢ : طبق وأضبط الخاصيتين Name ، Text على أداة زر أمر Button ؟

- قم بإنشاء Project جديد بإسم P22 .. تظهر الـForm1 .. أضف أداة زر أمر Button .. لاحظ أن قيمة الخاصية Name هي button1 قم بتعديلها وإعادة تسمية الأداة بالقيمة "btntitle" ... قم بضبط الخاصية Text برمجياً بمجرد تحميل الـForm ...
- في الحدث Load للـForm1 نكتب الكود بالشكل التالي :-

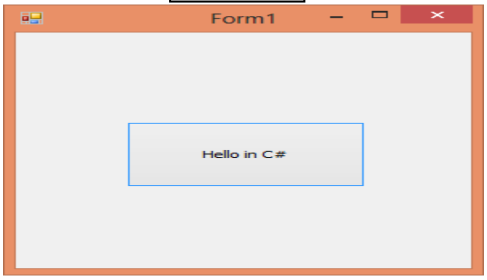
```
private void Form1_Load(object sender, EventArgs e)
        {
            btntitle.Text = "Hello in C#";
        }
```

- قم بتنفيذ البرنامج .. لاحظ شكل الزر قبل التنفيذ وبعد التنفيذ كما في الصور :

قبل التنفيذ



بعد التنفيذ



٣- الخاصية Backcolor : وتعني لون خلفية الأداة .. تستقبل وتعيد قيمة من نوع الصنف Color .. وتشترك فيها معظم عناصر التحكم .

٤- الخاصية Forecolor : وتعني لون عنوان الأداة أو اللون الأمامي للأداة .. تستقبل وتعيد قيمة نوع الصنف Color .. وتشترك فيها معظم عناصر التحكم .

- في التطبيق السابق (٢٢) أضف زري Button وأضبط الخصائص وفق الجدول التالي:-

الأداة	الخاصية Name	الخاصية Text
button1	btnbackcolor	لون خلفية الزر
button2	btnforecolor	لون عنوان الزر

- قم بإضافة أداة Colordialog (مربع حوار اللون) باسم colordialog1 ...
- في الحدث Click لـbtnbackcolor نكتب الكود بالشكل :-

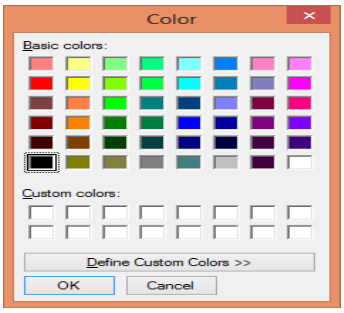
```
private void btnbackcolor_Click(object sender, EventArgs e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
                btntitle.BackColor = colorDialog1.Color;
        }
```

- في الحدث Click لـbtnforecolor نكتب الكود بالشكل :-

```
private void btnforecolor_Click(object sender, EventArgs e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
                btntitle.ForeColor = colorDialog1.Color;
        }
```

- قم بتنفيذ البرنامج .. إنقر على الزر " لون خلفية الزر " .. تظهر نافذة الألوان حدد اللون ثم أنقر Ok .. لاحظ تغير لون خلفية الزر btntitle .. إنقر على الزر " لون عنوان الزر " .. تظهر نافذة الألوان حدد اللون ثم إنقر Ok ... كرر العملية أكثر من مرة واستمتع .







٥- الخاصية Font : وتعني الخط لعنوان الأداة وتتضمن عدة خصائص للخط تتمثل في : نوع الخط ، حجم الخط ، نمط الخط ، تحته خط ، يتوسطه خط ، ...

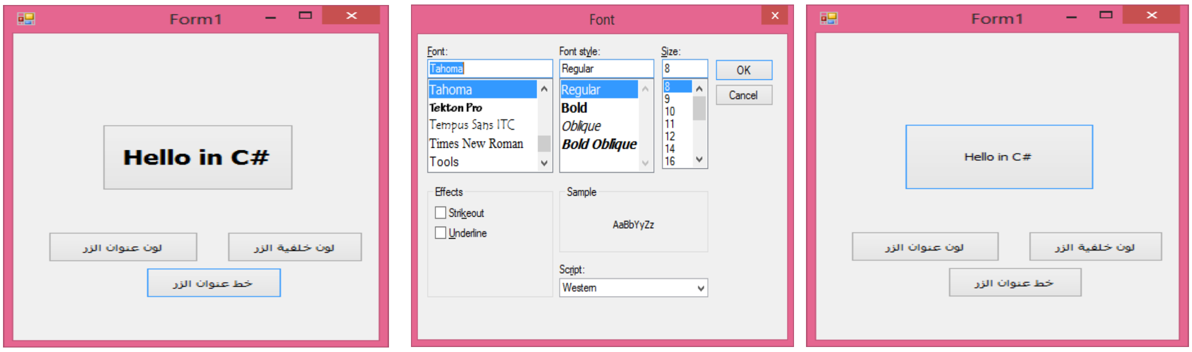
وتشترك في هذه الخاصية معظم عناصر التحكم .. تستقبل وتعيد قيمة من نوع الصنف font .

٦- الخاصية AutoSize : تقوم بإعادة تحجيم الأداة بما يتناسب مع حجم محتوياتها مثل حجم النص الموجود فيها .. تستقبل وتعيد قيمة من نوع Boolean إما True أو False .. جميع الأدوات إفتراضياً تحمل فيها الخاصية القيمة False ما عدا أداة العنوان Label فإن القيمة الإفتراضية له True بحيث يتغير حجم الـLabel بحجم النص الموجود فيه .

- في التطبيق السابق (٢٢) : قم بإجراء التعديلات التالية :
  - أضف زر Button وأضبط الخاصية Name بالقيمة "btnfont" واضبط الخاصية Text بالقيمة " خط عنوان الزر " ..
  - أضف أداة Fontdialog بإسم fontdialog1 ...
  - في الحدث Click للـbtnfont نكتب الكود بالشكل :-

```
private void btnfont_Click(object sender, EventArgs e)
        {
            btntitle.AutoSize = true;
            if (fontDialog1.ShowDialog() == DialogResult.OK)
                btntitle.Font = fontDialog1.Font;
        }
```

  - قم بتنفيذ البرنامج .. إنقر على الزر " خط عنوان الزر " .. تظهر نافذة الخط .. اضبط حجم الخط مثلاً ٢٤ .. لاحظ تغير حجم الخط وبالتالي يتغير حجم الزر btntitle ..كرر العملية أكثر من مرة ولاحظ تغير حجم الزر بتغير حجم النص الموجود بداخله ... إستمتع .



٧- الخاصية Enabled : وتعني أن الأداة تعمل أو لا .. تستقبل وتعيد قيمة من نوع Boolean .. في حالة كانت القيمة True يتم تفعيل أو تمكين الأداة وفي حالة كانت القيمة False يتم تعطيل الأداة .

- في التطبيق السابق (٢٢) : قم بإجراء التعديلات التالية :
  - قم بإضافة زر Button واضبط الخاصية Name بالقيمة "btnenabled" واضبط الخاصية Text بالقيمة "تعطيل الزر" .
  - في الحدث Click للـbtntitle نكتب الكود بالشكل :-

```
private void btntitle_Click(object sender, EventArgs e)
        {
            MessageBox.Show("لغة السي شارب تميز وإبداع ");
       }
```

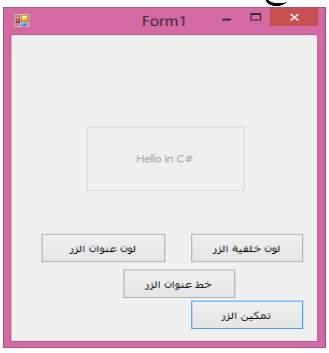
- في قسم التصريحات أعلى نافذة الـCode نكتب الكود كالتالي :-

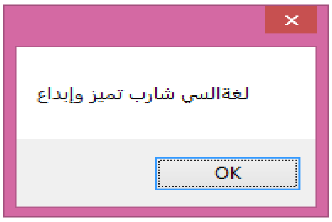
```
Boolean n1 = true;
```

- في الحدث Click للـbtnenabled نكتب الكود بالشكل :

```
private void btnenabled_Click(object sender, EventArgs e)
        {
            if (n1 == true)
            {
                n1 = false;
                btntitle.Enabled = false;
                btnenabled.Text = " تمكين الـزر ";
            }
            else
          {
                    n1 = true;
                    btntitle.Enabled = true;
                    btnenabled.Text = " تـعطيل الـزر  ";
               }
        }
```

- قم بتنفيذ البرنامج .. إنقر على الزر "Hello in C#" تظهر رسالة "لغة السي شارب تميز وإبداع" .. قم بالنقر على الزر "تعطيل الزر" .. حاول النقر على الزر "Hello in C#" مرة أخرى لاحظ تعطيل الزر .. إنقر الزر "تمكين الزر" .. لاحظ تفعيل الزر "Hello in C#" .. قم بالنقر عليه .. لاحظ ظهور الرسالة .. كرر العملية أكثر من مرة واستمتع ...







٨- الخاصية Visible : وتعني إظهار أو إخفاء الأداة .. تعيد وتستقبل قيمة من نوع Boolean في حال كانت القيمةTrue يعني إظهار الأداة أما إذا كانت القيمةFalse يعني إخفاء الأداة وتشترك فيها معظم عناصر التحكم .

- في التطبيق السابق(٢٢) : قم بإجراء التعديلات التالية:
  - أضف زر Button واضبط الخاصية Name بالقيمة "btnvisible" واضبط الخاصية Text بالقيمة "إخفاء الزر" ...
  - في قسم التصريحات أعلى نافذة الـCode نكتب الكود التالي :-

```
Boolean n2 = true;
```

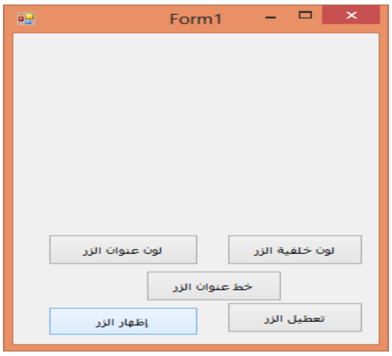
  - في الحدث Click للـbtnvisible نكتب الكود بالشكل :-

```
private void btnvisible_Click(object sender, EventArgs e)
        {
            if (n2 == true)
            {
                n2 = false;
                btntitle.Visible = false;
                btnvisible.Text = "إظهار الـزر ";
```

```
            }
            else
            {
                n2 = true;
                btntitle.Visible = true;
                btnvisible.Text = "إخفاء الزر ";
            }
        }
```

- قم بتنفيذ البرنامج .. إنقر على الزر " إخفاء الزر" .. لاحظ إختفاء الزر " Hello in C# " .. قم بالنقر على الزر "إظهار الزر" .. لاحظ ظهور الزر …





٩- الخاصية Cursor : تسمح بتغيير شكل مؤشر الفأرة أثناء مروره فوق الأداة المرتبطة بها .. ولها عدة أشكال .. إستعرض ذلك من نافذة الخصائص من الخاصية Cursor مثل ↕ أو ↔ … إلخ ذلك من أشكال مؤشر الماوس .. القيمة الإفتراضية لهذه الخاصية في جميع عناصر التحكم هي Default وتعني الشكل الإفتراضي لمؤشر الماوس وبالإمكان تغييرها كما سبق .. تستقبل وتعيد قيمة من Cursors …

- في التطبيق السابق (٢٢) : قم بإجراء التعديلات التالية:
  - في الحدث Load للـForm1 قم بإضافة الكود التالي :-

```
btntitle.Cursor = Cursors.NoMove2D;
```

  - قم بتنفيذ البرنامج .. حرك مؤشر الفأرة فوق الزر "Hello in C#" .. لاحظ تغير شكل مؤشر الفارة .

١٠- الخاصية Location : تعني موقع الأداة بالنسبة للـForm أو موقع الـ Form بالنسبة للشاشة.. وتحتوي على قيمتين من نوع int هما X محور السينات & Y محور الصادات داخل دالة البناء Point .. وتشترك فيها معظم عناصر التحكم .

١١- الخاصية Size : وتعني حجم الأداة وتحتوي على قيمتين من نوع int هما Width العرض ، Height الإرتفاع داخل دالة البناء Size .. وتشترك فيها معظم عناصر التحكم .

- تطبيق ٢٣ : صمم برنامجاً يستقبل ٤ قيم تضبط موقع وحجم الـForm ويتم إدخال القيم في مربعات نص ؟
  - قم بإنشاء Project جديد بإسم P23 .. تظهر الـForm1 .. قم بإضافة ٤ مربعات نص & ٤ ملصقات (أداة عنوان) & ٢ زر أمر وأضبط الخصائص كما في الجدول :-

إسم الأداة	الخاصية Name	الخاصية Text
label1	lblx	X
label2	lbly	Y
label3	lblwidth	Width
label4	lblheight	Height
textBox1	txtx	________
textBox2	txty	________
textBox3	txtwidth	________
textBox4	txtheight	________
button1	btnlocation	تغيير موقع النموذج
button2	btnsize	تغيير حجم النموذج

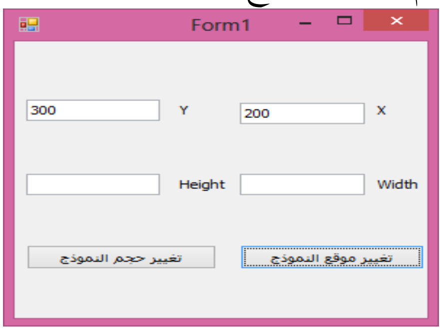
- في الحدث Click للـbtnlocation نكتب الكود بالشكل :-

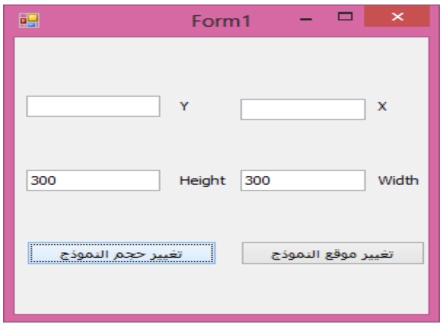
```
private void btnlocation_Click(object sender, EventArgs e)
        {
            int x = int.Parse(txtx.Text);
            int y = int.Parse(txty.Text);
            this.Location = new Point(x, y);
        }
```

- في الحدث Click للـbtnsize نكتب الكود بالشكل :-

```
private void btnsize_Click(object sender, EventArgs e)
        {
            int w = int.Parse(txtwidth.Text);
            int h = int.Parse(txtheight.Text);
            this.Size = new Size(w, h);
        }
```

- قم بتنفيذ البرنامج .. أضف قيم للـ X & Y ثم أنقر الزر "تغيير موقع النموذج" .. لاحظ تغير موقع الـForm بالنسبة للشاشة وفق القيم المدخلة .. أضف قيم للـ Width & Location ثم أنقر الزر "تغيير حجم النموذج" .. لاحظ تغير حجم الـForm وفق القيم المدخلة .. كرر العملية وأدخل قيم أخرى .. إستمتع .

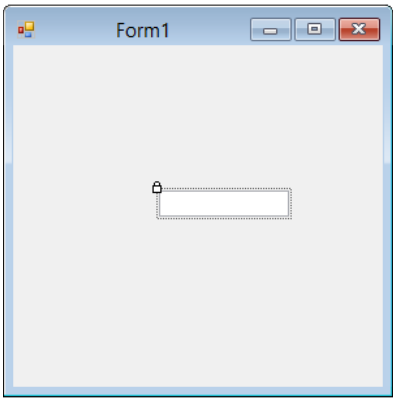




١٢- الخاصية Tag : تحوي بيانات تعريفية يكتبها المبرمج عن الأداة .. تستقبل وتعيد قيمة من نوع String .. وتشترك فيها معظم عناصر التحكم .. لكتابة معلومات عن مربع نص مثلاً برمجياً نكتب الكود بالشكل :-

```
txtx.Tag = " محور السينات ";
```

١٣- الخاصية Locked : تعني إقفال الأداة بحيث لا يمكن التحكم بأبعادها أو تغيير موقعها ويظهر القفل في مرحلة التصميم المرئي ويتم ضبط هذه الخاصية يدوياً فقط ولا يمكن ضبطها برمجياً .. تستقبل وتعيد قيمة من نوع Boolean .. في حال كانت Trueيتم إقفال الأداة أما في حال كانت False فإنه يتم تحرير أو فتح القفل لاحظ إقفال مربع النص كما في الصورة :-

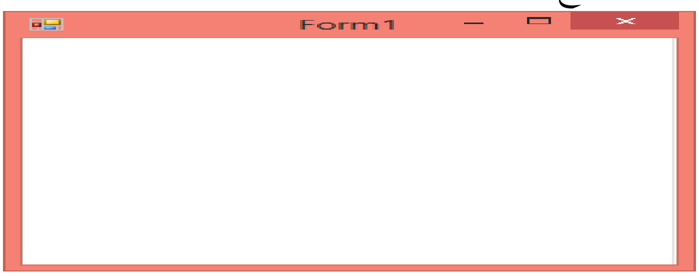


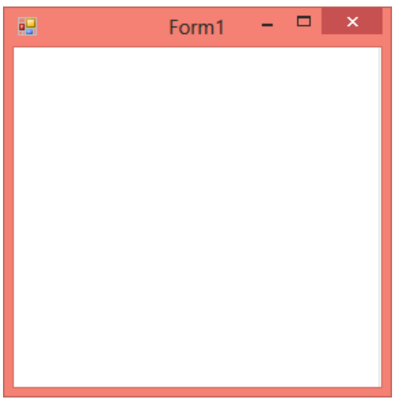
١٤- الخاصية Anchor : تدل على أي من حواف الأداة قدتم إرساؤها على الـForm .. تستقبل وتعيد قيمة من AnchorStyle وتحتمل عدة قيم كالتالي :

- None : بدون إرساء العنصر وهي القيمة الإفتراضية .
- Bottom : إرساء العنصر من الجهة السفلى .
- Top : إرساء العنصر من الجهة العليا .
- Left : إرساء العنصر من الجهة اليسرى .
- Right : إرساء العنصر من الجهة اليمنى .

ويمكن إرساء العنصر من أكثر من جهة .. تشترك معظم عناصر التحكم في هذه الخاصية .

- تطبيق ٢٤ : صمم برنامجاً يقوم بإرساء أداة RichRextBox من جميع الجهات الأربع يدوياً .. نفذ البرنامج .. قم بتغيير حجم الـForm .. ولاحظ ماذا يحدث ؟
  - قم بإنشاء Project جديد .. تظهر الـForm1 .. قم بإضافة أداة RichTextBox1 وأجعل حجمها بنفس حجم الـForm .. اضبط الخاصية Anchor للـrichTextBox1 من جميع الجهات الأربع .. قم بتنفيذ البرنامج .. أعد تحجيم الـForm من الحواف .. لاحظ أن حجم أداة RichTextBox تتغير بتغير الـForm .. كرر العملية واستمتع .





١٥- الخاصية Dock : تدل على أية حافة من حواف الأب يلتصق العنصر .. تستقبل وتعيد قيمة من نوع DockStyle وتحتمل عدة قيم كالتالي :

- None : عدم إلصاق العنصر على أي حافة من حواف الـForm .
- Fill : إلصاق العنصر على جميع حواف الـForm .
- Top : إلصاق العنصر على الحافة العليا للـForm .
- Bottom : إلصاق العنصر على الحافة السفلى للـForm .
- Left : إلصاق العنصر على الحافة اليسرى للـForm .
- Right : إلصاق العنصر على الحافة اليمنى للـForm .

وتشترك في هذه الخاصية معظم عناصر التحكم .

١٦- الخاصية AllowDrop : تحدد فيما إذا كان عنصر التحكم يمكن أن يقبل معطيات يتم إفلاتها عليه أم لا .. تستقبل وتعيد قيمة من Boolean إما True حينها تسمح الأداة بإفلات معطيات عليها أو False حينها لا تسمح بإفلات معطيات عليها .. وتوجد هذه الخاصية في بعض عناصر التحكم والتي تحوي معطيات بأنواعها ...

١٧- الخاصية BackgroundImage : تعني الصورة الخلفية للأداة .. تستقبل وتعيد قيمة من نوع الصنف Image والذي يمتلك طريقة FromFile والتي تحدد مسار وإسم وامتداد الصورة المطلوب جعلها صورة خلفية للأداة .. وتشترك في هذه الخاصية معظم عناصر التحكم والتي يمكن وضع خلفية صورة لها .

١٨- الخاصية RightToLeft : تحدد فيما إذا كانت محاذاة عنصر التحكم قد إنعكست ليقبل الخطوط من اليمين لليسار بحيث يتناسب مع الواجهات العربية .. تستقبل وتعيد قيمة من نوع RightToLeft وتحتمل عدة قيم كالتالي :

- Yes : محاذاة النص من اليمين إلى اليسار .
- No : تبقى محاذاة النص من اليسار إلى اليمين وهو الإفتراضي .
- Inherit : وتستخدم في حالة الوراثة .

١٩- الخاصية TabStop : تحدد ما إذا كان المستخدم يستطيع الوصول إلى عنصر التحكم باستخدام المفتاح Tab .. تستقبل وتعيد قيمة من نوع Boolean في حال كانت True يتم السماح بالإنتقال إلى الأداة بواسطة المفتاح Tab أو False يتم تعطيل الإنتقال بالمفتاح Tab .

٢٠- الخاصية Bottom : تمثل المسافة بين أسفل عنصر التحكم وقمة المنطقة التي تحتويه .. تستقبل وتعيد قيمة من نوع int .. وتشترك في هذه الخاصية معظم عناصر التحكم .

٢١- الخاصية Top : تمثل الإحداثية العلوية لعنصر التحكم .. تستقبل وتعيد قيمة من نوع int .. وتشترك في هذه الخاصية معظم عناصر التحكم .

٢٢- الخاصية Left : تمثل الإحداثية الأفقية للحافة اليسرى لعنصر التحكم مقاسة بوحدة الـPixel .. تستقبل وتعيد قيمة من نوع int .. وتشترك في هذه الخاصية معظم عناصر التحكم .

٢٣- الخاصية Right : تمثل المسافة بين الحافة اليمنى العليا لعنصر التحكم والحافة اليسرى لمحتويه .. تستقبل وتعيد قيمة من نوع int .. وتشترك في هذه الخاصية معظم عناصر التحكم .

// ملاحظة : توجد خصائص أخرى مشتركة بين معظم أو بعض عناصر التحكم سيتم تناولها ضمن سياق شرح الأدوات كلاً على حده في الفصول القادمة ...

الفصل العاشر

مربع النص TextBox &

مربع النص ذو التنسيق الغني RichTextBox &

مربع النص ذو القناع MaskedTextBox &

الملصقات (عناصر التسمية) Label &

عناصر الإرتباط LinkLabel